МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____Юрій КОНДРАТЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

СИСТЕМА ДЛЯ ГЕНЕРАЦІЇ ТА ПЕРЕВІРКИ ПАРОЛІВ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Валерій АРТИКОВ

«___»_____ 2025 р.

Керівник канд. техн. наук, доцент _____ Євген СІДЕНКО

«___»_____ 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп'ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп'ютерні науки
Освітня програма	Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Юрій КОНДРАТЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Артикова Валерія Степановича
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Система для генерації та перевірки паролів.

Керівник роботи: Сіденко Євген Вікторович, доцент кафедри інтелектуальних інформаційних систем, канд. техн. наук, доцент.

(прізвище, ім'я, по батькові, посада, науковий ступінь, вчене звання)

Затверджена наказом ЧНУ ім. Петра Могили від «18» листопада 2025 р. № 321.

2. Строк представлення кваліфікаційної роботи «___» червня 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: Розробка застосунку для перевірки та генерації паролів.

4. Перелік питань, що підлягають розробці: У кваліфікаційній роботі слід розробити теоретичні основи генерації та перевірки паролів, дослідити існуючі методи, сформувати вимоги до безпечних паролів, спроєктувати архітектуру системи, реалізувати алгоритми генерації й перевірки, забезпечити захист даних, провести тестування та сформулювати рекомендації.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ Євген СІДЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Валерій АРТИКОВ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «20» січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Система для генерації та перевірки паролів

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	18.12.2024	20.01.2025	Виконано
2	Аналіз предметної області та постановка задачі	21.01.2025	30.01.2025	Виконано
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема огляд проблем та рішень пов'язаних з темою.	31.01.2025	01.03.2025	Виконано
4	Огляд існуючих програмних рішень на ринку.	02.03.2025	01.04.2025	Виконано
5	Реалізація обраних технологій з аналізом отриманих результатів	02.04.2025	15.05.2025	Виконано
6	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	Виконано
7	Корегування роботи за результатами попереднього захисту	29.05.2025	29.05.2025	Виконано
8	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	Виконано
9	Доробка та остаточне оформлення КР	31.05.2025	10.05.2025	Виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	11.06.2025	11.06.2025	Виконано

Керівник роботи _____ Євген СІДЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Валерій АРТИКОВ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«20» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 402 ЧНУ ім. Петра Могили

Артикова Валерія Степановича

на тему: **“ СИСТЕМА ДЛЯ ГЕНЕРАЦІЇ ТА ПЕРЕВІРКИ ПАРОЛІВ ”**

Актуальність: у сучасному інформаційному суспільстві, де більшість особистих, фінансових, комерційних і державних процесів переходить у цифрову площину, питання інформаційної безпеки набуває особливої актуальності. Одним із базових і водночас найважливіших засобів захисту цифрових систем є паролі. Незважаючи на розвиток багатофакторної автентифікації, біометричних методів тощо, текстові паролі залишаються основним інструментом обмеження доступу в більшості систем.

Об’єктом роботи є процеси захисту персональних даних користувачів.

Предметом роботи виступають методи генерації, перевірки складності та хешування паролів.

Метою кваліфікаційної роботи є розробка вебзастосунку, що дозволяє користувачам створювати складні паролі та оцінювати їх стійкість до атак.

У роботі використано методи аналізу складності паролів, модульного проектування, розробки REST API, клієнт-серверної архітектури, сучасні бібліотеки мови Python, а також методи UI/UX-дизайну.

Кваліфікаційна робота складається зі вступу, трох розділів, висновків та списку використаних джерел. У першому розділі подано аналіз предметної області, виявлено проблему ненадійних паролів та розглянуто сучасні аналоги. У другому розділі описано програмну архітектуру майбутньої системи, компоненти, взаємодію модулів, засоби реалізації. Третій розділ присвячено реалізації функціоналу генерації паролів, перевірки їх складності та інтерфейсу користувача, тестування, приклади використання системи, а також оцінку ефективності.

Результатом роботи є програмний застосунок, що реалізує повний цикл роботи із паролями – від генерації до оцінки стійкості.

Кваліфікаційна робота містить: 78 сторінок, 5 таблиць, 31 рисунок, 2 додатки, 30 використаних джерел.

Ключові слова: *пароль, генерація, хешування, перевірка складності, інформаційна безпека, Python, REST API, вебзастосунок, інтерфейс користувача, авторизація*.

ABSTRACT

of the qualification work

of the applicant (s) of group 402 of Petro Mohyla Black Sea National University

Artikova Valeria Stepanovich

“SYSTEM FOR GENERATING AND VERIFYING PASSWORDS”

Relevance: in the modern information society, where most personal, financial, commercial and government processes are moving into the digital plane, the issue of information security is becoming particularly relevant.

The object of the work is the development of software in the field of information security.

The subject of the research is the methods of generation, complexity checking and hashing of passwords.

The purpose of the qualification work is to develop a web application that allows users to create complex passwords and assess their resistance to attacks.

The work uses methods of password complexity analysis, modular design, REST API development, client-server architecture, modern Python language libraries, as well as UI/UX design methods.

The first section presents an analysis of the subject area. The second section describes the software architecture of the future system, components, module interaction, and implementation tools. The third section is devoted to the implementation of the password generation functionality, checking their complexity and user interface, testing

The result of the work is a prototype of a software product that implements the full cycle of working with passwords - from generation to stability assessment.

The qualification work contains: 78 pages, 5 tables, 31 figures, 2 appendices, 30 sources used.

Keywords: password, generation, hashing, complexity check, information security, Python, REST API, web application, user interface, authorization.

ЗМІСТ

ВСТУП

У сучасному інформаційному суспільстві, де більшість особистих, фінансових, комерційних і державних процесів переходить у цифрову площину, питання інформаційної безпеки набуває особливої актуальності. Одним із базових і водночас найважливіших засобів захисту цифрових систем є паролі. Незважаючи на розвиток багатофакторної автентифікації, біометричних методів тощо, текстові паролі залишаються основним інструментом обмеження доступу в більшості систем.

Однак реальність свідчить про масове використання ненадійних паролів, повторне застосування одних і тих самих комбінацій, збереження паролів у незахищеному вигляді або на небезпечних носіях. Такі дії користувачів у поєднанні з прогалинами в технічному захисті призводять до значної кількості витоків даних та зловживань. Відповідно, виникає необхідність створення інструментів, які не лише полегшують процес створення складних паролів, але й перевіряють їхню надійність та гарантують захищене зберігання.

Об’єктом роботи є процеси захисту персональних даних користувачів.

Предметом роботи виступають методи генерації, перевірки складності та хешування паролів.

Метою кваліфікаційної роботи є розробка вебзастосунку, що дозволяє користувачам створювати складні паролі та оцінювати їх стійкість до атак.

Особливістю розробки є поєднання функціональності генератора, модуля аналізу складності, механізму хешування та сучасного адаптивного інтерфейсу користувача.

Для досягнення поставленої мети в роботі вирішуються такі основні завдання:

− проаналізувати актуальні загрози, пов’язані з використанням ненадійних паролів;

− дослідити існуючі рішення та визначити їхні переваги і недоліки;

− розробити функціональну архітектуру програмної системи;

− реалізувати основні компоненти генерації, перевірки та хешування паролів;

− створити інтерфейс користувача, адаптований до різних пристроїв;

− провести тестування системи та оцінити її ефективність.

Результатом кваліфікаційної роботи є повнофункціональний вебзастосунок, побудований на сучасних технологіях, який може бути використаний у якості самостійного сервісу або інтегрований у більші інформаційні системи для забезпечення високого рівня безпеки автентифікації користувачів.

РОЗДІЛ 1 АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО ГЕНЕРАЦІЇ ТА ПЕРЕВІРКИ ПАРОЛІВ

1.1 Аналіз предметної області

У сучасному цифровому середовищі питання захисту інформації набувають особливого значення. Інформаційні ресурси дедалі частіше стають об'єктами кібератак, спрямованих на отримання несанкціонованого доступу до конфіденційних даних. Одним з ключових елементів системи безпеки є автентифікація користувача, що найчастіше реалізується за допомогою паролів. Паролі відіграють роль першої лінії оборони між особистими або корпоративними даними та потенційним зловмисником.

Пароль – це комбінація символів, яка служить засобом підтвердження особи користувача в інформаційній системі. Його ефективність безпосередньо залежить від складності структури, довжини, унікальності та конфіденційності. Незважаючи на активне впровадження біометричних методів і багатофакторної автентифікації, паролі залишаються домінуючим засобом контролю доступу до більшості цифрових ресурсів. Варто зазначити, що жодна, навіть найдосконаліша технологія, не здатна повністю замінити або усунути необхідність використання паролів, що обумовлює їх ключову роль у сучасній моделі цифрової безпеки.

Серед основних загроз, які постають перед паролями, можна виділити кілька найпоширеніших і найбільш небезпечних:

− атаки перебором (brute force) – метод , при якому зловмисник автоматично перебирає всі можливі комбінації символів, поки не знайде правильний пароль. Така атака є особливо ефективною проти коротких або простих паролів, наприклад "123456", "password", "admin". У разі відсутності обмежень на кількість спроб або використання додаткового захисту (наприклад, блокування IP-адреси після кількох невдалих спроб) ці атаки можуть дати результати навіть за кілька хвилин;

− словникові атаки (dictionary attacks) – використовують заздалегідь підготовлені списки найбільш популярних паролів або слів, які часто

використовують люди. Цей метод значно швидший за перебір, і його ефективність базується на людській схильності до створення легко запам’ятовуваних, але слабких паролів. Наприклад, такі слова як "dragon", "football", "iloveyou" часто потрапляють у верхні рядки рейтингу найбільш вживаних паролів;

− фішинг – форма соціальної інженерії, при якій користувача обманом змушують розкрити свої паролі. Найчастіше фішинг реалізується через підроблені електронні листи або фальшиві вебсайти, які імітують інтерфейс відомих сервісів (банків, поштових служб, соцмереж). Ця атака не потребує технічного злому, але завдяки психологічному впливу може бути навіть ефективнішою;

− атаки із використанням витоків (credential stuffing) – зловмисники використовують бази даних зламаних акаунтів і намагаються застосувати ці ж логін-паролі на інших сайтах. Проблема у тому, що багато користувачів використовують одні й ті самі паролі на різних ресурсах. Такий підхід значно спрощує зловмисникам доступ до великої кількості облікових записів;

− шпигунське програмне забезпечення (кейлогери) – це програми, які непомітно встановлюються на пристрої користувача та записують усі натискання клавіш, включаючи паролі. Такі програми можуть передавати зібрану інформацію на сервери зловмисників. Іноді кейлогери маскуються під легітимні програми або додаються до піратського ПЗ;

− атаки через shoulder surfing – ситуація, коли зловмисник спостерігає за тим, як користувач вводить пароль, наприклад, у публічному місці або офісі. Особливо небезпечними є ситуації в транспорті, кафе, коворкінгах;

− окрім технічних методів злому, варто згадати ще й психологічні аспекти. Люди схильні використовувати прості паролі, пов'язані з особистою інформацією: датою народження, іменами близьких, улюбленими командами тощо. Це робить паролі вразливими до цілеспрямованих атак, зокрема соціальної інженерії. До того ж, багато користувачів не мають звички змінювати паролі регулярно, що ще більше підвищує ризик їхнього компрометування.

Додатково, поширеною проблемою є повторне використання паролів. Якщо один з них був скомпрометований, це автоматично створює загрозу для всіх інших облікових записів, до яких застосовується той самий пароль. Незважаючи на численні рекомендації щодо використання унікальних паролів для кожного сервісу, практика показує, що ця порада ігнорується значною частиною користувачів через зручність.

Для кращого розуміння рівня ризику необхідно враховувати не лише методи злому, а й масштаби наслідків. Витік одного пароля може призвести до втрати фінансових даних, блокування доступу до корпоративних ресурсів або повної компрометації цифрової особистості. Наприклад, доступ до електронної пошти може відкрити шлях до відновлення доступу до інших сервісів, що запускає ланцюгову реакцію втрати контролю над усім цифровим життям користувача.

Враховуючи вищезазначене, важливість створення ефективної, надійної та зручної системи управління паролями є беззаперечною. Така система має допомогти користувачу не лише створити надійний пароль, а й перевірити його на відповідність сучасним критеріям безпеки, попередити про потенційні ризики, а також забезпечити рекомендації для покращення захисту.

Таким чином, глибоке розуміння природи загроз, методів їх реалізації та поведінкових помилок користувачів дає змогу сформулювати вимоги до ефективної системи захисту паролів, яка не тільки технічно унеможливлює злам, а й виховує культуру безпечного поводження з конфіденційною інформацією. Питання захисту паролів не можна розглядати ізольовано – це частина більш широкої концепції цифрової гігієни та відповідального користування ресурсами Інтернету.

1.2 Аналіз існуючих методів генерації паролів

Генерація паролів є ключовим аспектом забезпечення інформаційної безпеки, оскільки саме надійність пароля визначає стійкість користувацьких акаунтів до різних кібератак. У сучасній практиці використовуються різні методи

генерації паролів, серед яких можна виділити три основні підходи: рандомна генерація, мнемонічна генерація та алгоритмічні методи. Кожен із них має свої переваги, недоліки й сфери застосування, тому їхній детальний аналіз є необхідним етапом у створенні надійної системи захисту.

Рандомна генерація паролів базується на випадковому виборі символів з певного набору, що може включати літери верхнього і нижнього регістрів, цифри та спеціальні символи. Найчастіше для цього використовуються генератори псевдовипадкових чисел (PRNG – Pseudo Random Number Generator), що реалізовані в мовах програмування (наприклад, функції *random()* у Python, *SecureRandom* у Java, або *crypto* в JavaScript). Основною перевагою цього методу є висока стійкість до атак перебором, оскільки паролі, згенеровані таким способом, зазвичай мають високий рівень ентропії та є непередбачуваними.

Водночас, рандомні паролі часто незручні для запам’ятовування, що змушує користувачів зберігати їх у небезпечних місцях (наприклад, у блокнотах або на робочому столі), або створювати простіші паролі для легшого запам’ятовування. Щоб зменшити ці ризики, сучасні сервіси рекомендують використовувати менеджери паролів (наприклад, Bitwarden, LastPass, 1Password), які можуть генерувати та безпечно зберігати складні паролі.

Мнемонічна генерація паролів полягає у створенні паролів, які користувач може легко запам'ятати, використовуючи певні фрази або логічні послідовності. Типовим прикладом такого методу є створення пароля з перших букв слів у легкозапам’ятовуваному реченні. Наприклад, фраза «У мене три коти і один пес» може бути перетворена на пароль «Ум3к1п». Для посилення безпеки додаються символи та великі літери: «Ум3К!1п». Деякі сервіси навіть дозволяють користувачам створювати свої мнемонічні шаблони для полегшення запам’ятовування.

Перевагою мнемонічних паролів є зручність для користувача, однак вони можуть бути вразливими, якщо створюються на основі популярних фраз або

особистої інформації. Зловмисники можуть використовувати соціальну інженерію або словникові бази даних, щоб спробувати підібрати подібні паролі.

Алгоритмічні методи генерації паролів передбачають створення паролів на основі певних математичних або логічних алгоритмів. Наприклад, пароль може формуватися на основі хешування введеної користувачем фрази або дати народження з використанням сольових значень для підвищення безпеки. Часто такі підходи реалізуються в корпоративних середовищах, де пароль генерується автоматично при реєстрації користувача в системі.

До переваг цього підходу належать автоматизація, стандартизація, контроль безпеки та масштабованість. Однак, якщо алгоритм стає відомим зловмиснику, паролі можуть бути передбачуваними. Тому важливим компонентом є використання криптографічно захищених методів генерації та зберігання паролів.

Сучасні приклади реалізації:

− Google пропонує вбудований генератор паролів у Chrome, який створює випадкові та надійні паролі;

− Apple у своїй системі iCloud Keychain також реалізує автоматичну генерацію та зберігання паролів;

− вебсервіси типу NordPass, Dashlane або KeePass дають можливість вибору типу генерації (рандомний, мнемонічний тощо), задаючи критерії довжини та складності.

У практиці безпечного програмування часто використовують бібліотеки з відкритим кодом, наприклад:

− *zxcvbn* від Dropbox – для перевірки стійкості паролів;

− *passlib* у Python – для генерації та перевірки паролів з криптографічними методами;

− *bcrypt*, *argon2*, *PBKDF2* – хешування паролів з сольовим значенням для додаткового захисту.

Таким чином, аналіз існуючих методів генерації паролів дозволяє визначити їхні сильні та слабкі сторони і сформувати основу для вибору оптимального

рішення під час проектування надійної системи генерації та управління паролями. Комбінування методів, адаптація під потреби користувача та інтеграція з сучасними інструментами є найбільш ефективним шляхом до підвищення безпеки цифрових систем.

1.3 Методи перевірки надійності паролів

Перевірка надійності паролів – це процес оцінювання рівня захищеності пароля від потенційних загроз, таких як атаки перебором, словникові атаки та соціальна інженерія. У сучасних інформаційних системах ця процедура є обов'язковим етапом при створенні або зміні пароля, оскільки дозволяє запобігти використанню слабких або компрометованих комбінацій.

Основна мета перевірки – гарантувати, що пароль достатньо складний і унікальний, щоб забезпечити захист облікового запису від несанкціонованого доступу. Існує кілька основних методів і підходів, які використовуються для перевірки надійності паролів. Нижче наведено таблицю (див. табл. 1.1) з коротким порівнянням ключових методів.

Таблиця 1.1 – Порівняння методів перевірки паролів.

№	Метод перевірки	Суть методу	Переваги	Недоліки
1	Оцінка складності пароля	Аналіз структури пароля (довжина, різні типи символів, відсутність повторів)	Простота реалізації, візуальна індентифікація	Може не враховувати зовнішні загрози
2	Оцінка ентропії	Визначення ступеня непередбачуваності пароля математичним шляхом	Висока точність при оцінці складності	Складність розуміння користувачем

Кінець таблиці 1.1

3	Чорні списки	Перевірка пароля у базах зламаних або найпоширеніших комбінацій	Захищає від використання вразливих паролів	Потребує регулярного оновлення баз
4	Машинне навчання	Аналіз поведінки, історії паролів, патернів створення	Адаптивність, гнучкість	Складність реалізації, потреба в даних
5	Перевірка витоків	Порівняння з відкритими базами компрометованих паролів	Дозволяє уникнути повторного використання зламаних даних	Залежність від сторонніх сервісів
6	Адаптивна перевірка	Варіативність складності паролів залежно від рівня доступу або ризику	Гнучке управління політиками безпеки	Необхідність правильної оцінки ризиків

Детальніше про основні терміни:

− складність пароля – показник, що відображає структуру пароля з точки зору різноманітності символів, довжини та унікальності. Наприклад, пароль "Pa$$w0rd123!" є складнішим за "password1";

− ентропія пароля – математичне очікування кількості інформації в паролі. Вища ентропія означає більшу кількість можливих комбінацій і складність злому. Розраховується за логарифмічною формулою, що враховує довжину пароля та обсяг символів;

− чорний список (blacklist) – список небажаних або небезпечних паролів, які були скомпрометовані або надто часто використовуються. Наприклад, паролі "123456", "qwerty", "admin";

− машинне навчання – підхід, при якому система вивчає патерни створення паролів, поведінкові особливості користувачів та адаптується до нових загроз. Наприклад, бібліотека *zxcvbn* прогнозує час, необхідний для злому пароля;

− сервіси перевірки витоків – онлайн-ресурси або API, які дозволяють перевірити, чи не був пароль скомпрометований. Наприклад, «Have I Been Pwned» дозволяє перевірити пароль без збереження введених даних;

− адаптивна перевірка – підхід, при якому вимоги до складності пароля варіюються залежно від рівня доступу користувача. Наприклад, для адміністратора система вимагатиме значно складніший пароль, ніж для звичайного користувача.

Приклади реалізації перевірки надійності:

− Google акаунти показують графічну оцінку складності під час введення пароля;

− GitHub блокує використання паролів із відкритих списків зламаних комбінацій;

− менеджери паролів (Bitwarden, LastPass) мають вбудовані інструменти для аналізу безпеки пароля.

Загалом, перевірка надійності паролів – це не лише технічне завдання, а й інструмент впливу на користувацьку поведінку. Вона сприяє підвищенню загального рівня інформаційної безпеки та формує культуру відповідального користування цифровими ресурсами.

1.4 Аналіз існуючих аналогів (програмних засобів)

У сучасному цифровому середовищі існує велика кількість програмних засобів, які реалізують функції генерації та перевірки паролів. Вони можуть бути реалізовані як у вигляді окремих додатків, так і як інтегровані компоненти веббраузерів або операційних систем. Аналіз таких рішень дозволяє оцінити їх функціональні можливості, ефективність, переваги та недоліки, а також визначити напрями для вдосконалення у рамках власної системи.

1. Менеджери паролів.

Це спеціалізовані програми, призначені для безпечного зберігання, генерації, організації та використання паролів. Вони працюють за принципом "один головний пароль – багато надійних паролів", тобто користувач має запам'ятати лише один головний пароль, щоб отримати доступ до всіх інших. Приклад відомих менеджерів паролів:

− NordPass – сучасний менеджер паролів від розробників NordVPN. Має кросплатформенну підтримку, шифрування на основі XChaCha20, автоматичне заповнення форм і вбудований сканер витоків;

− Dashlane – популярний комерційний сервіс, який окрім генерації та збереження паролів, також пропонує моніторинг даркнету, VPN, анонімну пошту для реєстрацій і автоматичну зміну паролів на підтримуваних сайтах;

− RoboForm – має простий інтерфейс, швидке заповнення форм, збереження облікових даних і можливість резервного копіювання. Підтримує синхронізацію між усіма пристроями користувача;

− інші популярні менеджери вже були описані раніше: LastPass, Bitwarden, 1Password, KeePass.

2. Вбудовані генератори у веббраузерах.

Сучасні браузери включають вбудовані інструменти для створення надійних паролів та їх збереження. Це зручно для користувачів, які не хочуть встановлювати окремі програми.

1. Google Chrome – пропонує автоматичне збереження і генерацію паролів із прив'язкою до Google-акаунта. Збережені паролі можна переглядати, редагувати та видаляти у Google Password Manager.

2. Mozilla Firefox – аналогічно має генератор паролів, моніторинг витоків через Firefox Monitor та збереження паролів у захищеному сховищі Lockwise.

3. Safari (Apple) – використовує iCloud Keychain для збереження, генерації та автозаповнення паролів. Паролі синхронізуються між усіма пристроями Apple.

3. Онлайн-сервіси генерації паролів.

Ці сервіси зазвичай не потребують реєстрації та дозволяють швидко створити надійний пароль відповідно до заданих параметрів:

- Norton Password Generator;
- Dashlane Password Generator;
- Strong Password Generator (strongpasswordgenerator.com).

Їхній недолік – відсутність збереження створених паролів, тому користувач має копіювати й зберігати їх самостійно.

4. Інструменти перевірки надійності паролів.

1. Zxcvbn (Dropbox) – JavaScript бібліотека для оцінки складності пароля. Аналізує шаблони, шаблонні послідовності, використовує евристику й бази популярних паролів. Широко використовується на сайтах для перевірки надійності пароля в реальному часі.

2. Have I Been Pwned – популярний онлайн-сервіс для перевірки, чи був пароль скомпрометований у витоку. Працює через API та інтегрується з багатьма менеджерами паролів.

3. Password Meter – простий вебінструмент, що показує графічну шкалу надійності на основі довжини пароля, різноманітності символів, наявності чисел та спецсимволів.

Порівняльна таблиця основних аналогів (див табл. 1.2)**.**

Загалом, аналіз існуючих програмних рішень свідчить про те, що успішні інструменти поєднують зручність користування, високу функціональність та сучасні методи перевірки надійності паролів. Водночас, значна частина рішень фокусується на окремих функціях – генерації або перевірці, тоді як інтегровані рішення залишаються більш ефективними в умовах повсякденного використання. Це дозволяє сформулювати вимоги до власної системи, враховуючи кращі практики існуючих аналогів.

Таблиця 1.2 – Порівняння відомих аналоогів

Назва	**Тип**	**Генерація**	**Перевірка на витоки**	**Візуальна оцінка**	**Відкритий код**
LastPass	Хмарний сервіс	+	+	+	–
Bitwarden	Хмарний/локальний	+	+	+	+
KeePass	Десктоп	+	–	–	+
1Password	Хмарний/корпоративний	+	+	+	–
NordPass	Хмарний	+	+	+	–
Dashlane	Хмарний	+	+	+	–
RoboForm	Хмарний/десктоп	+	–	+	–
Google Chrome	Вбудований	+	+ (через Google)	+	–
Firefox	Вбудований	+	+	+	–
zxcvbn	Бібліотека	–	–	+	+
Have I Been Pwned	Онлайн-сервіс	–	+	–	+

1.5 Постановка задачі

Аналіз предметної області, існуючих методів генерації та перевірки паролів, а також програмних аналогів показав необхідність створення універсальної, зручної та безпечної системи, яка забезпечувала б користувачам можливість генерувати надійні паролі та перевіряти їхню стійкість до зламу. У сучасних умовах стрімкого розвитку інформаційних технологій, зростання кількості кіберзагроз і

підвищення вимог до захисту персональних та корпоративних даних питання розробки інтелектуальних засобів для управління паролями є особливо актуальним.

Існуючі рішення, хоча й демонструють високий рівень функціональності, мають ряд обмежень: складність інтерфейсу, відсутність адаптації до користувацьких потреб, закритий вихідний код або ж недостатній рівень довіри до зберігання чутливої інформації у сторонніх сервісах. У зв’язку з цим у даній кваліфікаційній роботі ставиться мета створити інформаційну систему нового покоління, яка поєднуватиме функціональність, гнучкість, безпечність та простоту використання.

Об’єктом роботи є процеси захисту персональних даних користувачів.

Предметом роботи виступають методи генерації, перевірки складності та хешування паролів.

Метою кваліфікаційної роботи є розробка вебзастосунку, що дозволяє користувачам створювати складні паролі та оцінювати їх стійкість до атак.

Для досягнення поставленої мети в роботі вирішуються такі основні завдання:

− проаналізувати актуальні загрози, пов’язані з використанням ненадійних паролів;

− дослідити існуючі рішення та визначити їхні переваги і недоліки;

− розробити функціональну архітектуру програмної системи;

− реалізувати основні компоненти генерації, перевірки та хешування паролів;

− створити інтерфейс користувача, адаптований до різних пристроїв;

− провести тестування системи та оцінити її ефективність.

Обмеження та припущення:

− розробка передбачає використання лише відкритих або безкоштовних рішень, що дозволяє знизити витрати та забезпечити прозорість коду;

− основним користувачем системи розглядається приватна особа, студент, IT-фахівець або малий офіс, що потребує автономного рішення без збереження даних на сторонніх серверах;

− система реалізується у вигляді десктопного або вебдодатку, із можливістю автономного використання та потенційною підтримкою кросплатформенності.

Таким чином, сформульована задача охоплює як теоретичні, так і практичні аспекти побудови сучасного інструменту для безпечної роботи з паролями. Вона дозволяє комплексно вирішити проблему створення, перевірки, адаптації та покращення захисних механізмів у відповідності до сучасних стандартів інформаційної безпеки та користувацьких очікувань.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи було здійснено комплексний аналіз теоретичних та практичних аспектів, пов'язаних із проблематикою створення надійних паролів і забезпечення їхньої стійкості до різноманітних кіберзагроз. Зокрема, було детально розглянуто основні поняття предметної області, такі як автентифікація, ентропія, генерація паролів, перевірка їхньої надійності, а також типові загрози, пов’язані з використанням слабких або скомпрометованих паролів.

Проведено огляд сучасних підходів до генерації паролів, включаючи рандомні, мнемонічні та алгоритмічні методи. Розглянуто переваги та недоліки кожного з них, а також визначено, що ефективна система має використовувати комбінований підхід для забезпечення як безпеки, так і зручності для користувача.

У підпункті 1.3 описано сучасні методи перевірки надійності паролів, серед яких оцінка ентропії, використання чорних списків, машинне навчання та інтеграція з сервісами перевірки витоків. Це дозволяє формувати гнучку й адаптивну політику безпеки в інформаційних системах.

Окрему увагу було приділено аналізу існуючих програмних аналогів: менеджерів паролів, браузерних рішень, онлайн-генераторів та інструментів

перевірки. Було визначено, що найбільш ефективними є комплексні рішення, які об'єднують функції генерації, збереження, перевірки та рекомендацій щодо покращення паролів.

На основі проведеного аналізу було сформульовано мету кваліфікаційної роботи, визначено перелік конкретних задач, які необхідно вирішити в процесі проєктування та реалізації системи, а також зазначено ключові припущення і обмеження. Таким чином, розділ 1 закладає ґрунтовне теоретичне підґрунтя для подальшої практичної розробки інформаційної системи, яка забезпечуватиме надійне управління паролями та сприятиме підвищенню загального рівня кібербезпеки користувачів.

РОЗДІЛ 2 ПРОЕКТУВАННЯ СИСТЕМИ ГЕНЕРАЦІЇ ТА ПЕРЕВІРКИ ПАРОЛІВ

У цьому розділі буде представлено концептуальний підхід до побудови системи, що дозволяє не лише генерувати безпечні паролі, але й здійснювати комплексну перевірку їхньої надійності. Проєктування системи включає визначення вимог, вибір відповідних технологій, архітектури, а також розробку модулів, що відповідають за реалізацію функціоналу.

Метою цього етапу є побудова чіткої логічної структури програмного забезпечення, що поєднує інтерфейс користувача, серверну логіку та інструменти забезпечення безпеки. Така система має бути масштабованою, легко модифікованою та готовою до інтеграції з іншими інформаційними сервісами, якщо це буде потрібно в майбутньому.

2.1 Вибір засобів та інструментів розробки

Правильний вибір інструментів розробки є основою для побудови ефективної та безпечної інформаційної системи. З огляду на мету проєкту, необхідно забезпечити гнучкість реалізації, зручність розширення функціоналу, а також швидкість розробки і надійність у роботі.

Основні критерії вибору технологій:

- відкритість і вільна доступність;
- підтримка сучасних стандартів безпеки;
- можливість швидкої інтеграції модулів;
- простота розгортання на різних платформах;
- активна спільнота та багата документація.

Мова програмування Python – це високорівнева, інтерпретована мова програмування з відкритим кодом, яка поєднує простоту вивчення з потужним функціоналом. Вона широко використовується в різних галузях IT – від наукових досліджень і аналізу даних до веброзробки, автоматизації та кібербезпеки. Python

має зрозумілий синтаксис, який дозволяє розробникам створювати якісний код із мінімальними витратами часу.[1]

Однією з ключових переваг Python є його величезна стандартна бібліотека та безліч сторонніх модулів, які розширюють функціональність мови. У контексті створення системи генерації та перевірки паролів Python надає всі необхідні інструменти: генерацію випадкових значень, криптографічні функції, API для хешування паролів, обробку HTTP-запитів та взаємодію з базами даних.

Python підтримує об’єктно-орієнтоване, процедурне й функціональне програмування, що дозволяє будувати масштабовані та модульні системи. Завдяки активній спільноті Python постійно розвивається, має стабільну документацію і є стандартом де-факто в галузі розробки програмного забезпечення.

Мова також є кросплатформеною, тобто програми, написані на Python, можуть запускатися на будь-якій сучасній операційній системі без потреби в модифікаціях. Крім того, вона чудово інтегрується з іншими технологіями, має зручні засоби розгортання в хмарних середовищах і дозволяє швидко створювати прототипи систем.

Таким чином, Python є обґрунтованим вибором для розробки безпечної, адаптивної та зручної у використанні системи генерації та перевірки паролів.

Нижче представлені інші інструменти (детальний огляд):

− Flask – мікрофреймворк на мові Python, що дозволяє швидко створити вебзастосунок з мінімальним налаштуванням. Flask ідеально підходить для створення легких, але функціональних систем. Підтримує шаблонізатор Jinja2, що дозволяє генерувати HTML на сервері. Завдяки модульній архітектурі легко масштабувати або інтегрувати додаткові модулі без суттєвих змін у коді;

− FastAPI – сучасний асинхронний вебфреймворк для Python, який дозволяє створювати API високої продуктивності. Має вбудовану підтримку OpenAPI та автоматичну генерацію інтерактивної документації, що є надзвичайно зручним при розробці та тестуванні. FastAPI має високу швидкість роботи та відмінно підходить для проєктів, де передбачається значне навантаження;[2]

− HTML/CSS/JavaScript + Jinja2 або React – для створення інтерфейсу користувача. Jinja2 – зручний шаблонізатор, що інтегрується з Flask, дозволяє рендерити сторінки на сервері з мінімальними витратами. React – JavaScript бібліотека для побудови інтерфейсів користувача. Застосовується для створення динамічних односторінкових застосунків (SPA) та інтерактивного UX. React дозволяє реалізувати більш складну логіку на клієнтській стороні, зокрема перевірку введених паролів у реальному часі;

− zxcvbn – бібліотека, яка надає більш «людяний» підхід до перевірки складності паролів. Вона враховує повторення, патерни, комбінації, схожі на популярні фрази чи особисті дані. Результатом є не просто цифрова оцінка, а й підказки для користувача щодо покращення пароля;[3]

− passlib – набір криптографічних алгоритмів, що дозволяє хешувати паролі з використанням сучасних методів, таких як bcrypt, PBKDF2, scrypt та argon2. Підтримує перевірку паролів на відповідність хешу, автоматичне оновлення хешів у разі зміни алгоритму тощо;[10]

− secrets – стандартна бібліотека Python, що забезпечує генерацію випадкових значень, які підходять для використання в криптографічних цілях. Вона дозволяє створювати паролі, ключі, токени з високим рівнем ентропії;[4]

− SQLite – вбудована реляційна база даних, яка не вимагає окремого сервера. Ідеально підходить для невеликих локальних застосунків або прототипів;

− PostgreSQL – потужна об’єктно-реляційна СУБД, яка підтримує складні запити, великі обсяги даних, розширення, а також засоби для забезпечення безпеки на рівні бази даних. Використовується в продуктивних системах із високим навантаженням.

2.2 Архітектура та структура системи

2.2.1 Інтерфейс користувача (Frontend)

Інтерфейс користувача є критично важливою складовою будь-якої інформаційної системи, що орієнтована на кінцевого користувача. Саме він

визначає перше враження про програму, зручність її використання, інтуїтивність, доступність функцій і загальний комфорт взаємодії. У системі генерації та перевірки паролів інтерфейс має на меті надати користувачу зручні інструменти для введення даних (наприклад, бажаної довжини пароля, типів символів), активації генерації, перевірки введеного пароля, а також перегляду рекомендацій щодо його зміцнення.

Розробка інтерфейсу базується на таких принципах:

− простота та мінімалізм – користувач не повинен відволікатися на зайві елементи. Кожна деталь інтерфейсу має мати конкретне призначення та не перевантажувати простір;

− інтуїтивність – всі кнопки та поля повинні бути логічно розташовані, мати зрозумілі позначення та не викликати труднощів у користуванні навіть для людей без технічного досвіду;

− миттєвий зворотний зв’язок – після введення пароля одразу ж має з’явитися оцінка його складності, що дозволяє користувачу оперативно скорегувати свій вибір;

− доступність – інтерфейс повинен бути адаптованим під різні пристрої (десктопи, планшети, мобільні телефони), враховуючи гнучкість розмітки, адаптивну сітку та сенсорну навігацію;

− безпека – жодні паролі не повинні зберігатися або передаватися без згоди користувача. Усі дії, що можуть вплинути на конфіденційність, мають супроводжуватися відповідними попередженнями.

Реалізація інтерфейсу користувача може здійснюватися кількома способами, залежно від обраного підходу до архітектури:

− класичний HTML/CSS/JS з використанням шаблонізатора Jinja2 (у разі вибору Flask як серверного фреймворку). Такий підхід дозволяє створити прості HTML-сторінки, які динамічно формуються сервером. Він зручний при невеликій кількості функцій і забезпечує швидке завантаження сторінок без складної клієнтської логіки. Jinja2 дозволяє вбудовувати змінні та керуючі конструкції

(умови, цикли) безпосередньо в HTML-код, що значно спрощує відображення динамічних даних;

− учасний фреймворк на кшталт React – дає змогу створити динамічний інтерфейс користувача у форматі SPA (Single Page Application). Це означає, що всі зміни на сторінці (наприклад, перевірка складності пароля, відображення рекомендацій тощо) відбуваються без повного перезавантаження, завдяки чому взаємодія з системою стає значно швидшою і приємнішою для користувача. React також дозволяє зручно розділяти інтерфейс на незалежні компоненти, що полегшує його підтримку й тестування.

До типових елементів інтерфейсу в такій системі належать:

− поля введення параметрів пароля (довжина, символи, складність);

− перемикачі та чекбокси для вибору опцій (включення цифр, спеціальних символів, букв у верхньому/нижньому регістрі);

− кнопка "Згенерувати пароль", що ініціює запит до сервера;

− поле з результатом генерації, яке автоматично оновлюється без перезавантаження сторінки;

− інтерактивна шкала надійності пароля, реалізована у вигляді кольорової шкали або смуги прогресу;

− текстові підказки та рекомендації, що формуються в залежності від складності пароля;

− кнопка "Копіювати в буфер обміну" для зручного збереження пароля користувачем;

− кнопка "Очистити форму" або "Змінити параметри", що скидає вибрані налаштування до типових значень.

Додатково можуть бути реалізовані функції:

− темна/світла тема оформлення;

− багатомовна підтримка (наприклад, англійська/українська);

− анімації під час відображення результатів (напр., плавна поява рекомендацій).

На основі отриманих від користувача параметрів, інтерфейс формує запити до серверної частини системи. Завдяки цьому чіткому розподілу обов'язків frontend відповідає лише за взаємодію, а логіка перевірки та генерації залишається безпечною на сервері. Така структура підвищує надійність та дає змогу масштабувати систему в майбутньому – наприклад, додати мобільну версію або браузерне розширення.

2.2.2 Серверна логіка (Backend)

Серверна логіка є основою всієї функціональності системи, оскільки саме вона відповідає за обробку запитів від користувача, виклик необхідних алгоритмів, взаємодію з базою даних та повернення відповідей у зручному для відображення форматі. У контексті системи генерації та перевірки паролів цей компонент виконує всі операції, що стосуються обчислень, перевірки введених параметрів, логування активностей, захисту від атак і забезпечення коректного функціонування. Це критичний компонент, що забезпечує правильну роботу всієї системи. Від її надійності, безпеки та продуктивності залежить якість обслуговування користувачів. Чітка структуризація, використання перевірених фреймворків, реалізація обробки помилок та безпечна обробка даних – усе це є запорукою стабільного функціонування сервісу.

Серверну логіку рекомендовано реалізувати за принципом багаторівневої архітектури:

− рівень маршрутизації (routes) – відповідає за обробку запитів користувача та передачу їх у відповідні контролери. Приклад: /generate, /check, /history, /api/docs;

− контролери (views/controllers) – обробляють бізнес-логіку: приймають параметри, викликають відповідні сервіси, формують відповіді;

− сервіси (services) – виконують конкретні обчислювальні задачі: генерацію паролів, перевірку надійності, хешування;

− модулі доступу до даних (data access) – забезпечують збереження та читання з бази даних;

− система логування (logging) – реєструє ключові події, помилки, спроби несанкціонованого доступу.

Такий підхід забезпечує чіткий розподіл обов’язків між частинами коду, спрощує супровід, покращує читабельність та дозволяє масштабувати систему.

Для реалізації серверної логіки доцільно використовувати мову **Python** завдяки її простоті, широкій підтримці бібліотек, високій швидкості розробки та читабельності коду.

Flask:

− легкий фреймворк, який дозволяє швидко реалізувати REST API;

− підходить для навчальних і малих проєктів;

− добре інтегрується з шаблонізатором Jinja2;

− має багато сторонніх розширень.

FastAPI:

− асинхронний фреймворк нового покоління;

− підтримка автоматичної генерації Swagger UI;

− працює швидко і ефективно навіть при великій кількості одночасних запитів;

− має вбудовану перевірку типів і валідацію через Pydantic.

Ключові функції backend-модуля:

− прийом запитів від користувача Наприклад, POST-запит із параметрами генерації пароля: довжина, символи, складність. Дані передаються у форматі JSON або як параметри форми;

− передача даних у відповідні модулі В залежності від типу запиту дані надходять у функцію генерації або перевірки;

− генерація відповіді Сервер повертає результат (наприклад, згенерований пароль або оцінку його складності) у форматі JSON, який потім візуалізується frontend-компонентом;

− робота з базою даних (опційно) Якщо реалізовано функцію збереження історії перевірок, створених паролів, налаштувань користувача – запит спрямовується до бази (наприклад, SQLite або PostgreSQL);

− логування та аудит Усі запити, помилки, винятки реєструються у лог-файлах. Це необхідно для діагностики, виявлення атак або збоїв;

Захист і обмеження:

− захист від XSS, SQL-ін’єкцій і CSRF;

− обмеження кількості запитів на одиницю часу (rate limiting);

− шифрування паролів при передачі (використання HTTPS);

− очистка та валідація вхідних параметрів.

2.2.3 Блок генерації паролів

Блок генерації паролів є одним із найважливіших і центральних модулів системи, оскільки саме він виконує одну з базових задач – створення надійного, випадкового та криптографічно стійкого пароля, який задовольняє як потреби користувача, так і сучасні стандарти безпеки. Генерація паролів – це не просто випадкове поєднання символів, а продуманий процес, який враховує цілу низку параметрів, обмежень і варіантів використання. Важливо також розуміти, що якісний пароль є першою лінією оборони від зовнішніх загроз, тому цей модуль повинен бути реалізований з максимальною увагою до безпеки, гнучкості та зручності використання.

У сучасному цифровому середовищі, де практично всі сервіси потребують аутентифікації, значення якісної генерації паролів стає дедалі актуальнішим. Використання слабких паролів призводить до численних витоків даних, атак типу "brute-force" і соціальної інженерії. Саме тому завдання генератора – не лише надати функціональний результат, а й сформувати культуру безпечного поводження з обліковими даними. Блок генерації повинен слугувати не тільки технічним засобом, а й навчальним елементом.

Роль генератора в архітектурі системи. Блок генерації паролів є частиною логіки на серверному рівні та тісно інтегрується з інтерфейсом користувача (frontend), а також із модулем перевірки складності паролів. Він отримує параметри генерації з форми введення, виконує обчислення на основі криптографічних методів і повертає результат у вигляді строки пароля. Також передбачено можливість взаємодії з базою даних (наприклад, для збереження шаблонів або налаштувань користувача).

Крім того, генератор може бути реалізований як окремий мікросервіс або API-інтерфейс, що дозволяє використовувати його в інших системах, наприклад у корпоративних платформах, хмарних сервісах або мобільних додатках. Така архітектура забезпечує масштабованість і гнучкість розгортання системи.

Розширення параметрів та їх значення. Користувач має змогу самостійно задати конфігурацію майбутнього пароля. Це важливо як з точки зору персоналізації, так і для відповідності корпоративним політикам безпеки. Наприклад:

− встановити довжину пароля в діапазоні від 8 до 64 символів;

− додати обов’язкову наявність хоча б одного символа кожного типу (цифри, літери, спецсимволи);

− заборонити використання повторів символів або послідовностей на кшталт "abc", "123";

− вказати шаблон або маску для генерації (наприклад, формат типу "XX-999-ZZ");

− вибрати набір допустимих символів або виключити ті, що можуть бути некоректно оброблені (лапки, слеші, пробіли).

Також система може підтримувати профілі генерації (наприклад: «простий», «рекомендований», «максимальна безпека»), які автоматично заповнюють параметри згідно з типом користувача або обраною політикою. Ці профілі можуть бути пов’язані з корпоративними вимогами або конкретними технічними стандартами, що забезпечує відповідність регламентам безпеки.

Пояснення щодо генерації. Ключовою особливістю генератора є використання надійного джерела ентропії. Саме тому застосовується бібліотека secrets (а не random), яка забезпечує необхідний рівень непередбачуваності. Генерація символів відбувається не просто випадково, а з урахуванням вказаних умов. Для перевірки коректності алгоритму можуть бути використані unit-тести, що перевіряють довжину, склад, шаблон, унікальність, відсутність заборонених символів тощо.

Окрім того, у майбутньому можливе розширення до генерації паролів на основі штучного інтелекту – наприклад, створення фраз із високим рівнем запам’ятовуваності, але складних з точки зору автоматизованого зламу. Такі підходи вже використовуються в деяких сучасних системах і базуються на обробці природної мови (NLP), аналізі патернів безпечних комбінацій та адаптивному навчанні на основі зворотного зв’язку від користувачів.

Система має також супроводжувати процес генерації поясненнями, наприклад:

- "Додано 3 спеціальних символи для підвищення стійкості";
- "Згенерований пароль не містить повторів символів";
- "Уникнуто послідовності літер і цифр";
- "Застосовано профіль "Максимальна безпека" – пароль містить 16 символів, усі категорії, відсутні словникові збіги".

Ці рекомендації допомагають користувачу краще зрозуміти логіку генератора, а також служать своєрідним навчальним компонентом, який формує навички безпечного цифрового поводження. Це сприяє підвищенню загального рівня обізнаності щодо кіберзагроз та мінімізує ризик повторного використання слабких паролів.

Нижче представлені сценарії використання

Генератор паролів може застосовуватися не лише для створення паролів доступу. Можливе його використання в таких випадках:

- генерація API-ключів і токенів авторизації;

− тимчасові паролі або токени доступу (OTP) з обмеженим терміном дії;

− секретні коди активації продуктів або ліцензій;

− генерація кодів для багатофакторної автентифікації (2FA);

− генерація паролів для локальних облікових записів у автономних системах (наприклад, IoT-пристрої).

Таким чином, модуль має бути достатньо гнучким і універсальним, щоб задовольнити не лише базові потреби користувачів, а й особливі сценарії використання у корпоративних, адміністративних чи навіть індустріальних середовищах. Його можна адаптувати для інтеграції з великими інформаційними системами або платформами віддаленого керування.

Блок генерації паролів – це не просто інструмент випадкового набору символів. Це критичний компонент, який формує перший і найбільш важливий рівень безпеки в цифровій системі. Його розробка повинна враховувати широкий спектр функціональних можливостей, сценаріїв використання, параметризації, гнучкості та відповідності стандартам. Розумний, захищений і зручний у використанні генератор є невід'ємною складовою сучасних систем автентифікації й захисту даних. Без належної реалізації цього модуля будь-яка система ідентифікації залишається вразливою до найпоширеніших загроз – від атаки словником до повного компрометування бази користувачів.[15]

2.2.4 Блок перевірки надійності паролів

Блок перевірки надійності паролів – це один з ключових функціональних елементів системи, який відіграє вирішальну роль у забезпеченні безпеки користувацьких облікових записів. Його основне завдання полягає у визначенні ступеня надійності заданого пароля, тобто оцінці ймовірності його вгадування або зламу різними способами, включно з атаками типу "brute-force", словниковими атаками та методами соціальної інженерії. Від якості роботи цього блоку безпосередньо залежить рівень захисту системи та обізнаність користувача щодо слабких місць у його вибраному паролі.

У сучасному інформаційному середовищі, де витоки паролів стали регулярним явищем, особливу цінність має надання негайного зворотного зв’язку користувачу щодо надійності обраного пароля. Такий підхід не лише допомагає сформувати більш захищену поведінку в мережі, а й знижує ймовірність зламу облікових записів через використання слабких або повторно використаних паролів. Регулярне використання інструментів оцінки дозволяє виробити у користувача звичку самостійно перевіряти якість паролів, перш ніж використовувати їх у важливих облікових записах.

Оцінка пароля здійснюється за допомогою спеціалізованих бібліотек, зокрема zxcvbn, розробленої компанією Dropbox. Цей інструмент використовує евристичні методи для виявлення передбачуваних шаблонів, поширених замін символів (наприклад, "P@ssw0rd"), особистих даних, словникових слів, а також бере до уваги довжину, різноманітність символів і контекст пароля.

Крім zxcvbn, можуть також використовуватись інші аналітичні підходи, наприклад:

- перевірка паролів на наявність у базах зламаних паролів;
- підрахунок ентропії – математичного показника непередбачуваності пароля;
- визначення шаблонів: літерні, числові, змішані, тощо;
- виявлення легко вгадуваних елементів, таких як клавішні послідовності або імена користувача.

Такі методи підвищують точність оцінювання, дозволяючи системі гнучко адаптуватися до нових загроз у кіберпросторі.

Кожен пароль оцінюється за шкалою (зазвичай від 0 до 4):

- **0** — дуже слабкий (легко вгадується миттєво);
- **1** — слабкий (може бути вгаданий у межах кількох секунд);
- **2** — помірний (потребує кількох хвилин або годин для злому);
- **3** — надійний (злам потребує значного часу та ресурсів);

– **4** — дуже надійний (практично не піддається злому за поточних технологій).

Додаткові критерії оцінювання:

– використання лише одного типу символів (наприклад, тільки літери);

– повторення символів (наприклад, "аааааа");

– послідовності клавіш на клавіатурі ("qwerty", "asdfgh");

– використання імен, прізвищ, дат народження, популярних фраз;

– збіг із найпоширенішими паролями з баз витоків;

– простота введення – якщо пароль легко набрати на клавіатурі, він, швидше за все, буде простим для злому.

Система повинна:

– автоматично аналізувати пароль при кожному введенні в полі форми;

– відображати поточний рівень безпеки у вигляді кольорової індикації (наприклад, червоний – слабкий, зелений – надійний);

– виводити текстову оцінку з поясненнями ("Довжина пароля достатня, але слід додати спецсимволи");

– надавати конкретні рекомендації щодо покращення (наприклад, "Додайте символи верхнього регістру", "Збільште довжину пароля");

– бути гнучкою до адаптації під корпоративні політики, дозволяти зміну ваги критеріїв;

– уникати збереження перевірених паролів, забезпечуючи повну конфіденційність.

Інтерфейс повинен бути достатньо зрозумілим і простим, щоб навіть користувач без технічних знань міг зрозуміти, чому його пароль оцінюється саме так, а також що можна зробити для його покращення.

Освітня функція. Окрім суто технічної перевірки, блок перевірки повинен мати навчальний компонент. У реальному часі система пояснює, чому саме певний пароль є небезпечним, і як його можна покращити. Це допомагає користувачу не просто змінити пароль в конкретному випадку, а й надалі формувати сильніші

паролі самостійно, навіть без допомоги системи. Таким чином, користувач підвищує свою цифрову грамотність і стає менш уразливим до соціальної інженерії та фішингових атак. Крім того, можна реалізувати інтерактивні поради, приклади гарних і поганих паролів, візуалізацію довжини, унікальності, схожості до типових шаблонів тощо. У майбутньому такий модуль може перетворитися на окрему освітню платформу з оцінками, рівнями складності, тестами.

Нижче наведені перспективи розвитку застосунку

Можливе вдосконалення модуля:

− інтеграція з базами зламаних паролів (наприклад, HaveIBeenPwned API) для порівняння пароля з відомими витоками;

− аналіз контексту (чи пароль схожий на e-mail користувача, ім’я, адресу);

− додаткова градація складності з урахуванням поведінкових даних, частоти зміни паролів;

− вивід середнього часу на злам пароля згідно з поточними обчислювальними можливостями;

− підтримка багатомовних словників та локалізованих рекомендацій;

− синхронізація з іншими модулями системи для формування єдиного звіту про безпеку;

− запам’ятовування останніх оцінок паролів (на локальному пристрої) для відстеження прогресу користувача;

− підключення нейромереж для адаптивного аналізу нових загроз і методів створення паролів;

− звіти про стан безпеки паролів (особливо для корпоративних користувачів);

− динамічний аналіз зломів або підозрілих змін паролів.

Приклади паролів із поясненням. Щоб користувачі краще розуміли, як працює система оцінки, і які саме паролі вважаються небезпечними або надійними, доцільно навести практичні приклади (див. табл. 2.1).

Таблиця 2.1 – Приклади та оцінка паролів

Пароль	**Оцінка**	**Пояснення**
123456	0	Один із найпоширеніших паролів. Повністю передбачуваний.
qwerty	0	Послідовність клавіш на клавіатурі. Легко вгадується.
password	0	Типове словникове слово. Дуже часто зустрічається в зламах.
P@ssw0rd	1	Заміна символів стандартна, легко розпізнається системами зламу.
Mariia1999	1–2	Можливе особисте ім’я + рік народження. Легко зламати методом перебору.
8w*D#rK!	3	Хороша комбінація символів, але короткий (8 символів).
LgH#97!cKvW2@z	4	Довгий, випадковий, включає всі типи символів. Висока складність.
sunshineCLOUD99!	3–4	Мнемонічний пароль з комбінацією слів, цифр і символів. Добре запам’ятовується.

Ці приклади можуть бути інтерактивно вбудовані у систему як підказки або відображатись при аналізі конкретного введеного пароля.

2.2.5 Блок хешування та зберігання паролів

Один із найважливіших аспектів безпеки будь-якої системи автентифікації – це правильне зберігання паролів. Навіть найнадійніший пароль втрачає свою цінність, якщо він зберігається у відкритому вигляді або недостатньо захищеному форматі. Блок хешування та зберігання паролів має забезпечити криптографічно стійкий метод обробки паролів, який унеможливлює їхнє зчитування або відновлення в разі витоку бази даних.

Зберігати паролі у відкритому вигляді категорично заборонено. Натомість слід зберігати лише їх криптографічний хеш. Хеш-функція – це одностороння функція, яка перетворює вхідні дані (у нашому випадку пароль) у фіксовану послідовність байтів або символів. Навіть незначна зміна вхідного значення призводить до повністю іншого хешу. Важливо, що ця операція є незворотною: за хешем неможливо відновити початковий пароль.

Хешування також дозволяє здійснювати автентифікацію без потреби зберігати чи передавати справжній пароль. Коли користувач вводить пароль, система хешує його й порівнює з тим, що збережено у базі даних. Якщо хеші збігаються – автентифікація успішна.

Використання алгоритмів хешування. Існує кілька поширених алгоритмів, які використовуються для безпечного хешування паролів:

− **bcrypt** – популярний алгоритм, який автоматично додає соль (random salt) та дозволяє налаштувати складність через параметр "cost". Завдяки цьому, час обчислення хешу зростає експоненційно зі збільшенням складності, ускладнюючи масові атаки. Добре підтримується у Python через бібліотеки bcrypt і passlib;

− **PBKDF2** – алгоритм, що реалізує ітераційне хешування, використовуючи HMAC і задану кількість ітерацій. Перевагою є підтримка на рівні стандартних бібліотек (наприклад, hashlib). Часто використовується в корпоративному програмному забезпеченні;

− **scrypt** – створений для протидії атакам з використанням апаратного забезпечення. Потребує значного обсягу оперативної пам’яті, що робить його менш зручним для зловмисників із великими ресурсами;

− **argon2** – сучасний алгоритм, який вважається найбільш стійким і надійним. Підтримує паралельні обчислення, параметри споживання пам’яті та часу. Алгоритм активно використовується в системах з підвищеним рівнем безпеки.

Соль та її значення. Для кожного пароля необхідно генерувати унікальну **соль** – випадкову послідовність байтів, яка додається до пароля перед обчисленням

хешу. Це запобігає використанню попередньо обчислених таблиць (rainbow tables) і гарантує, що однакові паролі в системі будуть мати різні хеші.

У сучасних реалізаціях соль додається автоматично і зберігається разом із хешем у зашифрованому вигляді. Деякі алгоритми, як-от bcrypt або argon2, кодують соль безпосередньо у фінальному хеші, що полегшує верифікацію пароля.

Реалізація в Python. Для реалізації безпечного зберігання паролів у системі доцільно використовувати бібліотеку *passlib*, яка підтримує всі основні алгоритми хешування та має простий і зрозумілий API (див. рис. 2.1).

```python
from passlib.hash import bcrypt

# Хешування пароля
hashed_password = bcrypt.hash("MySecurePassword123")

# Перевірка
if bcrypt.verify("MySecurePassword123", hashed_password):
    print("Password is valid")
```

Рисунок 2.1 – Використання бібліотеки *passlib*

Також підтримуються інші алгоритми через *passlib.context.CryptContext*, що дозволяє централізовано задавати політику безпеки (див. рис. 2.2).

```python
from passlib.context import CryptContext

pwd_context = CryptContext(schemes=["argon2", "pbkdf2_sha256"], deprecated="auto")

hash = pwd_context.hash("MySecurePassword123")
if pwd_context.verify("MySecurePassword123", hash):
    print("Access granted")
```

Рисунок 2.2 – Алгоритм через *passlib.context.CryptContext*

2.2.6 База даних

Система зберігання та обробки даних є основою інформаційної системи, і саме база даних (БД) відіграє ключову роль у забезпеченні цілісності, доступності

й організованого зберігання всієї необхідної інформації. У контексті системи генерації та перевірки паролів БД використовується для збереження налаштувань користувачів, історії генерації, хешованих паролів, логів дій, сеансів, звітів про надійність паролів, шаблонів генерації та навіть параметрів безпеки самої системи. База даних у системі перевірки паролів – це не просто сховище інформації. Вона є критичним вузлом взаємодії, без якого не можливе збереження, контроль та аналіз даних. Саме від її структури, безпеки, масштабованості й правильної інтеграції з іншими модулями залежить ефективність і надійність усієї системи. Ретельно спроєктована й оптимізована БД – запорука стабільної та захищеної цифрової інфраструктури.

База даних має забезпечити:

- швидке зчитування й запис інформації (з низькою латентністю);
- структуроване зберігання хешованих паролів та історії користування;
- підтримку транзакцій для забезпечення цілісності записів (ACID-властивості);
- гнучке керування структурою даних (можливість додавання нових типів записів);
- можливість горизонтального масштабування (для великих проєктів);
- інструменти для резервного копіювання та аварійного відновлення;
- підтримку шифрування даних на рівні таблиць, полів або всього сховища;
- детальний розподіл доступу за ролями та правами користувачів (RBAC);
- журналювання доступу та дій (audit logging) з захистом від фальсифікацій.

Для локального або тестового середовища можна використовувати SQLite – просту в налаштуванні і легку систему. Для промислового використання – PostgreSQL, MySQL або інші реляційні СУБД, які підтримують усі потрібні функції.

Висновки до розділу 2

У другому розділі було розглянуто архітектуру, основні компоненти та технічні засоби реалізації системи для генерації та перевірки паролів. Кожен підпункт демонстрував ключові аспекти проектування, взаємодії модулів, вибору технологій та підходів до забезпечення зручності й безпеки користувача.

Особливу увагу приділено мові програмування Python, яка завдяки своїй гнучкості та широкому набору бібліотек дозволяє швидко створювати надійні рішення. Було розкрито переваги використання фреймворків і бібліотек для обробки REST API, хешування, створення інтерфейсу користувача та реалізації бази даних. Усі ці компоненти тісно взаємодіють між собою, створюючи єдиний функціональний продукт, що відповідає вимогам сучасної інформаційної безпеки.

Значну частину уваги зосереджено на забезпеченні зручності користувача та адаптивності інтерфейсу. Детально описано вимоги до візуальної складової системи, її доступності, багатомовності та підтримки різних платформ.

Таким чином, другий розділ заклав технічну основу майбутньої реалізації системи. Усі обґрунтовані рішення – від вибору технологій до структурної організації – сприятимуть ефективному виконанню поставлених задач та реалізації функціонального, надійного й масштабованого програмного продукту.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Реалізація генератора паролів

Реалізація програмного забезпечення здійснювалася з використанням мови програмування **Python** у поєднанні з фреймворком **FastAPI**. Такий вибір обумовлений високою швидкістю розробки, зручністю побудови REST API та широкою підтримкою бібліотек, необхідних для роботи з безпекою, криптографією та аналізом текстових даних.

Серверна частина. Для створення серверної частини застосовано FastAPI, що забезпечує асинхронну обробку запитів, автоматичну генерацію OpenAPI-документації (Swagger UI), а також підтримку перевірки та серіалізації вхідних даних за допомогою бібліотеки pydantic.

Файл main.py містить ініціалізацію FastAPI-додатку та підключення окремих маршрутів:

```
from fastapi import FastAPI
from backend.generator import router as gen_router
from backend.strength import router as strength_router

app = FastAPI()

app.include_router(gen_router, prefix="/generate")
app.include_router(strength_router, prefix="/strength")
```

Нижче на рисунку 3.1 наведено результат запуску ервера FastAPI у середовищі розробки.

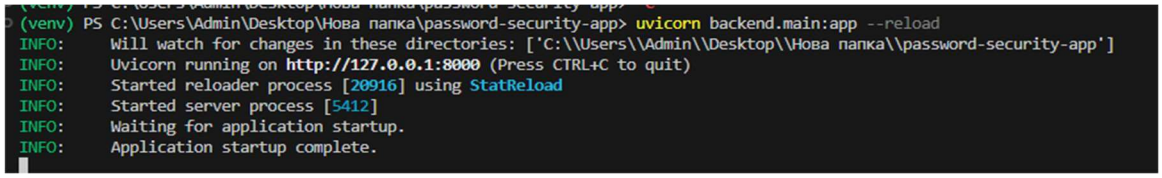


Рисунок 3.1 – Результат запуску сервера

Клієнтська частина. Інтерфейс користувача створено за допомогою **HTML5**, **CSS3** та **JavaScript**, без залучення сторонніх фреймворків. Такий підхід дозволив зосередитися на мінімалізмі, адаптивності та контролі над функціональністю.

Файл index.html містить структуру інтерфейсу з формами для введення параметрів пароля, перемикачами та кнопками (рис.3.2).

```
  <input type="number" id="length" value="12" min="4" max="65">
</label>
<small class="hint">Порада: 12–20 символів — оптимально для безпеки</small><br>
<label><input type="checkbox" id="upper" checked> Додавати великі літери</label><br>
<label><input type="checkbox" id="digits" checked> Додавати цифри</label><br>
<label><input type="checkbox" id="specials" checked> Додавати спецсимволи</label><br>
<label><input type="checkbox" id="easy"> Легко запам'ятовуваний пароль</label><br>
<button id="generateBtn">Згенерувати</button>
```

Рисунок 3.2 – Структура інтерфейсу

Клієнтська логіка реалізована у файлі **app.js**, де описано обробку подій, надсилання запитів до API, відображення результатів, перемикання темної теми та мови (рис. 3.3).

```
document.getElementById("themeToggle").addEventListener("click", () => {
  document.body.classList.toggle("dark-mode");
  const isDark = document.body.classList.contains("dark-mode");
  document.getElementById("themeToggle").textContent = isDark ? "🌞" : "🌙";
});

document.getElementById("langToggle").addEventListener("click", () => {
  const current = localStorage.getItem("lang") || "ua";
  const newLang = current === "ua" ? "en" : "ua";
  localStorage.setItem("lang", newLang);
  applyLanguage(newLang);
});
```

Рисунок 3.3 – Перемикачі теми сайту та мови

Нижче наведений фрагмент інтерфейсу користувача (рис. 3.4).

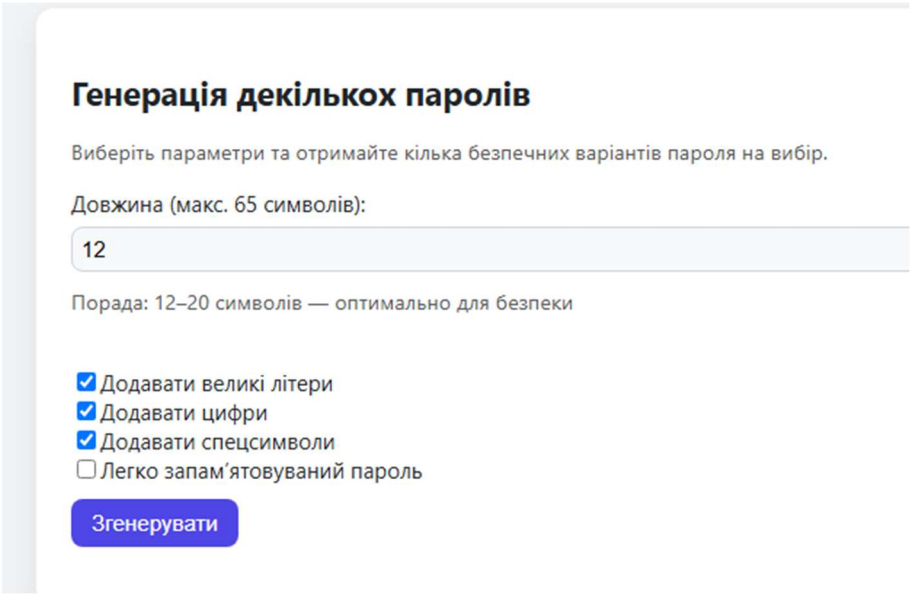


Рисунок 3.4 – Фрагмент основного інтерфейсу користувача

Нижче наведений результат роботи застосунку після натискання на клавішу генерувати (рис. 3.5).

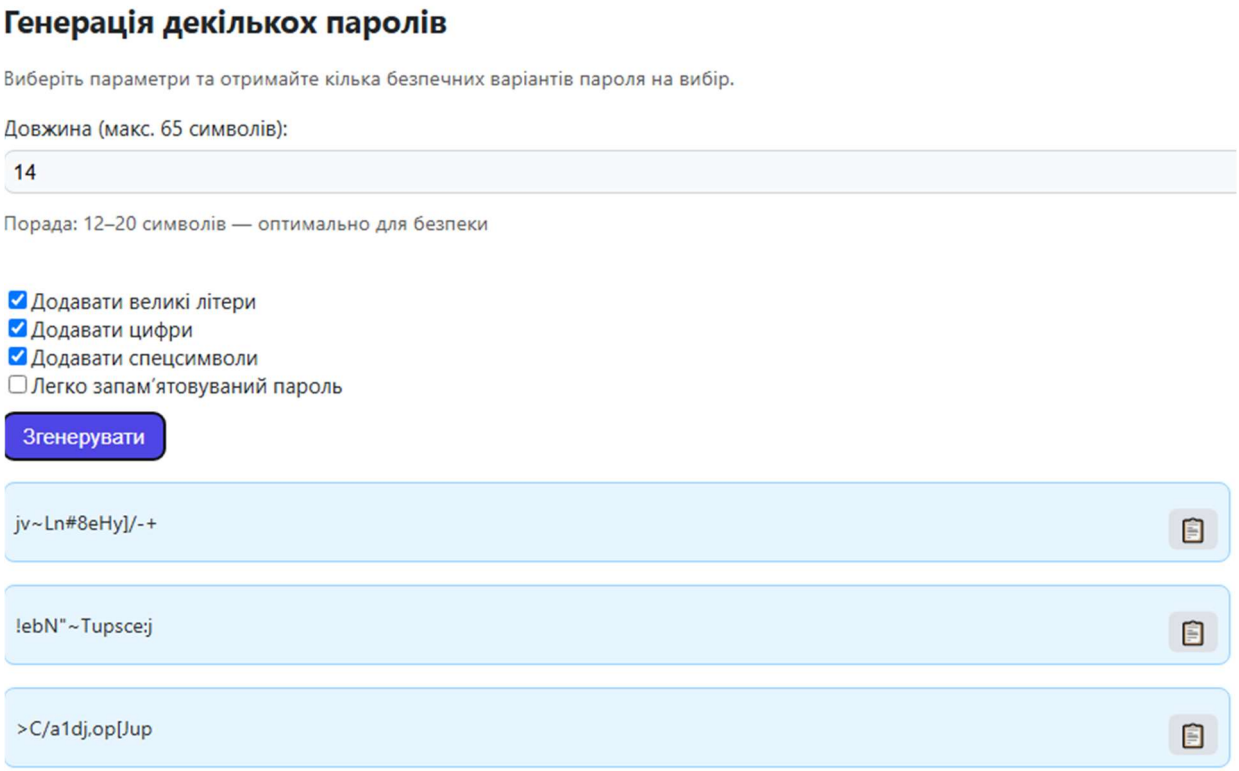


Рисунок 3.5 – Результат

Для організації проєкту застосовано:

- віртуальне середовище **venv** – для ізоляції залежностей;
- редактор коду **Visual Studio Code** – як універсальне середовище для розробки;
- **Uvicorn** – для локального запуску **FastAPI**;
- **Swagger UI** – для тестування **API** через вбудовану документацію;
- **zxcvbn** – для перевірки складності паролів;
- **requests** – для взаємодії з **API** перевірки паролів у витоках;
- **secrets** – для криптографічної генерації символів;
- **Chart.js** – (на етапі розширення) для візуалізації складності.

На рисунку зображено фрагмент коду для генерації паролів (рис. 3.6).

```
document.getElementById("generateBtn").addEventListener("click", async () => {
  const length = Math.min(parseInt(document.getElementById("length").value), 65);
  const use_upper = document.getElementById("upper").checked;
  const use_digits = document.getElementById("digits").checked;
  const use_specials = document.getElementById("specials").checked;
  const easy = document.getElementById("easy").checked;

  const passwordOptions = document.getElementById("passwordOptions");
  passwordOptions.innerHTML = "";

  for (let i = 0; i < 6; i++) {
    try {
      const res = await fetch(`${API_BASE}/generate/`, {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          length,
          use_upper,
          use_digits,
          use_specials,
          easy_to_remember: easy
        }),
      });
```

Рисунок 3.6 – Функція генерації паролю

Пояснення параметрів зображених на рисунку 3.6 наведені в таблиці 3.1.

Таблиця 3.1 – Пояснення параметрів

Параметр	Тип	Опис
length	number	Довжина пароля (обмежується максимумом у 65 символів)
use_upper	boolean	Якщо true, включаються великі літери (A–Z)
use_digits	boolean	Якщо true, включаються цифри (0–9)
use_specials	boolean	Якщо true, додаються спеціальні символи (!@#$% тощо)
easy_to_remember	boolean	Якщо true, генерується "запам’ятовуваний" пароль, наприклад Blue-Tiger42!

3.2 Архітектура програмної системи

Структура програмної системи побудована за принципом чіткого поділу клієнтської та серверної частин, що дозволяє ефективно масштабувати застосунок, тестувати окремі модулі й легко оновлювати інтерфейс незалежно від API.

Програмна система реалізована як односторінковий застосунок (SPA), де фронтенд взаємодіє із сервером через REST API. Всі дані передаються у форматі JSON. На рисунку 3.7 зображена повна структура проєкту.

```
PASSWORD-SECURITY-APP/
├── backend/
│   ├── generator.py        ← генерація паролів
│   ├── strength.py         ← перевірка складності + HIBP
│   ├── schemas.py          ← моделі для валідації вхідних даних
│   └── main.py             ← запуск FastAPI-сервера, підключення маршрутів
│
├── frontend/
│   ├── index.html          ← структура інтерфейсу
│   ├── styles.css          ← стилі світлої та темної тем
│   └── app.js              ← логіка взаємодії з API
│
```

Рисунок 3.7 – Загальна структура проєкту

Клієнт-серверна модель

Взаємодія між частинами відбувається за протоколом HTTP через REST API. З боку фронтенду запити надсилаються за допомогою fetch, а сервер відповідає у форматі JSON (див. табл. 3.2).

Таблиця 3.2 – Взаємодія через REST API

Метод	Маршрут	Призначення
POST	/generate/	Генерація пароля
POST	/strength/	Аналіз складності + перевірка витоків

Файл **main.py**: точка входу (рис. 3.8).

```
app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
app.include_router(generator_router, prefix="/generate")
app.include_router(strength_router, prefix="/strength")
```

Рисунок 3.8 – Реалізація входу через main.py

Файл **generator.py** – генерація паролів

Обробка запиту /generate/, генерація відбувається залежно від параметрів:

– стандартна (випадкові символи);

– легко запам’ятовувана (Red-Fox23@).

Код функції (рис. 3.9).

```
def generate_strong_password(length, use_upper, use_digits, use_specials):
    charset = string.ascii_lowercase
    if use_upper:
        charset += string.ascii_uppercase
    if use_digits:
        charset += string.digits
    if use_specials:
        charset += string.punctuation
    return ''.join(random.choice(charset) for _ in range(length))
```

Рисунок 3.9 – Код функції

Файл **strength.py** – перевірка складності

Використовується бібліотека **zxcvbn**, яка повертає оцінку score (0–4) і рекомендації (рис. 3.10).

```python
@router.post("/")
def check_password(data: PasswordCheckRequest):
    result = zxcvbn(data.password)
    score = result['score']
    feedback = result['feedback']['suggestions']
    pwned = check_password_pwned(data.password)
    return {
        "score": score,
        "feedback": feedback,
        "breached": pwned
    }
```

Рисунок 3.10 – Використання бібліотеки **zxcvb**

Паралельно відбувається перевірка пароля через API сервісу Have I Been Pwned (HIBP) за допомогою протоколу k-Anonymity.

Фронтенд: **app.js** + **index.html**

Користувач взаємодіє з інтерфейсом через форми, перемикачі, кнопки. Всі запити йдуть на локальний сервер (рис. 3.11).

```javascript
const res = await fetch(`${API_BASE}/generate/`, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({
    length,
    use_upper,
    use_digits,
    use_specials,
    easy_to_remember: easy
  }),
});
```

Рисунок 3.11 – Функція взаємодії користувача з застосунком

3.3 Реалізація генератора паролів

Генерація паролів є основною функцією системи. Реалізація охоплює створення як звичайних криптографічно стійких паролів, так і

легкозапам’ятовуваних комбінацій, що відповідають сучасним вимогам зручності й безпеки.

API-маршрут

Генерація виконується через маршрут POST /generate/. У тілі запиту (формат JSON) надсилаються такі параметри (рис. 3.12).

```
class PasswordParams(BaseModel):
    length: int = 12
    use_upper: bool = True
    use_digits: bool = True
    use_specials: bool = True
    easy_to_remember: bool = False
```

Рисунок 3.12 – Параметри тіла запиту

Параметри:

- length – довжина пароля;
- use_upper – чи включати великі літери;
- use_digits – чи включати цифри;
- use_specials – чи включати спецсимволи;
- easy_to_remember – прапорець для генерації варіанту, що легко запам’ятовується.

Серверна реалізація генерації (**generator.py**)

Модуль **generator.py** обробляє запити до API, аналізує параметри та викликає відповідну функцію генерації (рис. 3.13).

```
def generate_strong_password(length, use_upper, use_digits, use_specials):
    charset = string.ascii_lowercase
    if use_upper:
        charset += string.ascii_uppercase
    if use_digits:
        charset += string.digits
    if use_specials:
        charset += string.punctuation
    return ''.join(random.choice(charset) for _ in range(length))
```

Рисунок 3.13 - Звичайний генератор

Символи обираються з відповідного пулу (**charset**) випадковим чином. Для генерації використовується модуль **secrets** – рекомендований для криптографічної стійкості.

На рисунку 3.14 представлений код для легко запам’ятовуваного варіанту пароля.

```
adjectives = ["Red", "Blue", "Green", "Fast", "Bright", "Happy", "Dark", "Swift"]
animals = ["Fox", "Tiger", "Wolf", "Bear", "Lion", "Eagle", "Shark", "Panther"]
specials = "!@#$%&*"

def generate_easy_password(use_digits=True, use_specials=True):
    word = f"{random.choice(adjectives)}-{random.choice(animals)}"
    number = str(random.randint(10, 99)) if use_digits else ""
    special = random.choice(specials) if use_specials else ""
    return f"{word}{number}{special}"
```

Рисунок 3.14 – Реалізація модуля легко запам’ятовуваного пароля

Нижче наведений результат роботи модуля (рис. 3.15).

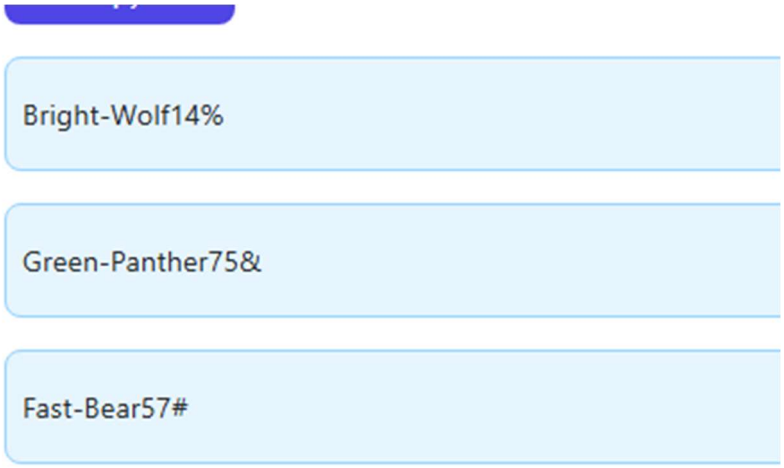


Рисунок 3.15 – Результат

Обробка на клієнтській стороні (**app.js**)

Користувач взаємодіє з UI через форму з параметрами. Дані зчитуються через DOM-елементи й надсилаються на сервер (рис. 3.16).

```
const res = await fetch(`${API_BASE}/generate/`, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({
    length,
    use_upper,
    use_digits,
    use_specials,
    easy_to_remember: easy
  }),
});
```

Рисунок 3.16 - Обробка на клієнтській стороні

Отриманий пароль виводиться на екран у списку можливих варіантів (рис. 3.17).

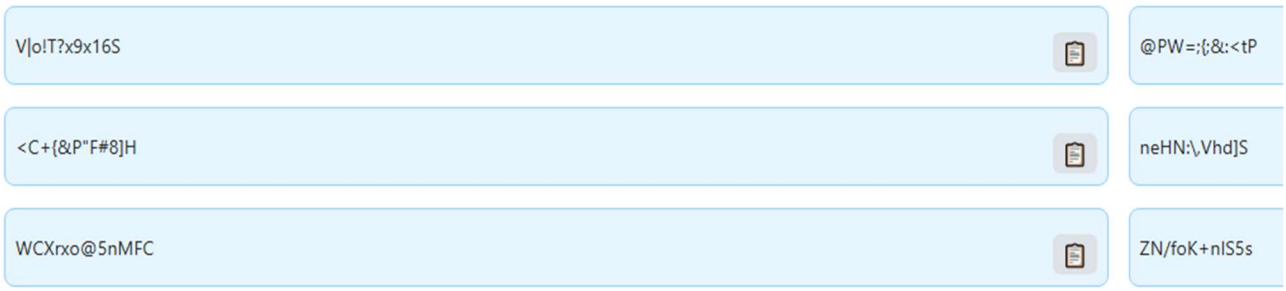


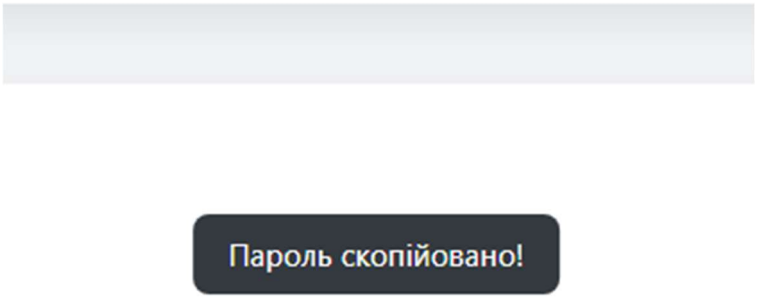
Рисунок 3. 17 – Декілька можливих результатів генерації

UI-функціональність:

– після генерації пароль відображається одразу;

– кожен варіант має кнопку "📋" для копіювання в буфер обміну;

– підтвердження копіювання показується у вигляді toast-повідомлення (рис. 3.18).



Ртсунок 3.18 - Toast-повідомлення при копіюванні

Переваги реалізації:

– висока швидкість генерації (від 1 до 6 варіантів одночасно);

– можливість кастомізації: довжина, типи символів, простота запам’ятовування;

– повна інтеграція з інтерфейсом користувача без перезавантаження сторінки.

3.4 Перевірка складності пароля

Модуль перевірки складності пароля реалізовано як окремий компонент серверної частини застосунку. Його метою є оцінка криптографічної стійкості пароля на основі кількісних та якісних характеристик, а також перевірка на наявність у загальнодоступних витоках паролів.

Аналіз складності: бібліотека zxcvbn

Для аналізу використовується бібліотека **zxcvbn**, розроблена компанією Dropbox. Вона дозволяє оцінювати надійність пароля не лише за його довжиною, а й з урахуванням:

- повторюваних символів;
- відомих шаблонів (123456, qwerty, password);
- імен користувачів, електронних адрес тощо;
- наявності послідовностей клавіш;
- частоти використання в реальних витоках.

Нижче представлений фрагмент коду функції strength.py (рис. 3.19).

```python
@router.post("/")
def check_password(data: PasswordCheckRequest):
    result = zxcvbn(data.password)
    score = result['score']
    feedback = result['feedback']['suggestions']
    pwned = check_password_pwned(data.password)
    return {
        "score": score,
        "feedback": feedback,
        "breached": pwned
    }
```

Рисунок 3.19 – Функція для перевірки паролю

Перевірка пароля на наявність у витоках (HIBP)

Окрім аналізу структури пароля, додано перевірку на наявність його в реальних витоках даних, використовуючи сервіс Have I Been Pwned. Перевірка здійснюється за допомогою алгоритму k-Anonymity, що не розкриває сам пароль.

Нижче наведений фрагмент коду для запиту до HIBP (рис. 3.20).

```
def check_password_pwned(password: str) -> str:
    sha1_password = hashlib.sha1(password.encode('utf-8')).hexdigest().upper()
    prefix = sha1_password[:5]
    suffix = sha1_password[5:]

    url = f"https://api.pwnedpasswords.com/range/{prefix}"

    try:
        response = requests.get(url, timeout=5)
        if response.status_code != 200:
            return "⚠ Помилка перевірки через HIBP"

        for line in response.text.splitlines():
            hash_suffix, count = line.strip().split(":")
            if hash_suffix == suffix:
                return f"⚠ Цей пароль зустрічався {count} раз(ів) у витоках"
        return "✅ Пароль не знайдено у зламаних базах"
    except Exception:
        return "❌ Не вдалося з'єднатися з HIBP"
```

Рисунок 3.20 – Фрагмент коду для запиту до HIBP

Результат роботи функції у застосунку (рис. 3.21).

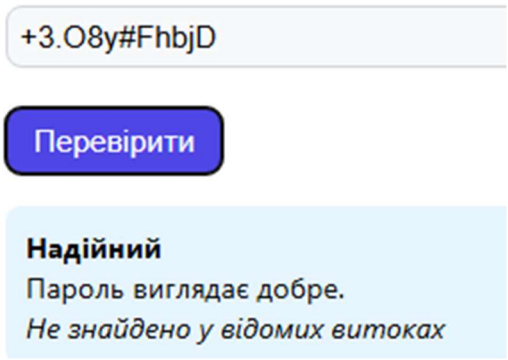


Рисунок 3.21 – Результат перевірки пароля

Відображення результатів на клієнтській частині

У файлі app.js функція checkBtn обробляє відповідь API та формує результат для виводу (рис. 3.22).

```
const strengthMessages = {
  ua: ["Дуже слабкий", "Слабкий", "Посередній", "Добрий", "Надійний"],
  en: ["Very weak", "Weak", "Medium", "Good", "Strong"]
};

const fallbackMsg = {
  ua: "Пароль виглядає добре.",
  en: "The password looks good."
};

let breach = "";
if (data.breached?.error) {
  breach = lang === "ua"
    ? "Помилка з'єднання з базою витоків"
    : "Error connecting to breach database";
} else if (data.breached?.breached) {
  breach = lang === "ua"
    ? `Знайдено у витоках (${data.breached.count} разів)`
    : `Found in breaches (${data.breached.count} times)`;
} else {
  breach = lang === "ua"
    ? "Не знайдено у відомих витоках"
    : "Not found in known breaches";
}

const title = strengthMessages[lang][data.score];
const feedback = data.feedback.length ? data.feedback.join("<br>") : fallbackMsg[lang];
```

Рисунок 3.22 – Фрагмент коду для обробки відповідей

Нижче наведений скріншот результату перевірки в інтерфейсі з рейтингом, порадами і позначкою "Знайдено у витоках (X разів)" (рис. 3.23).

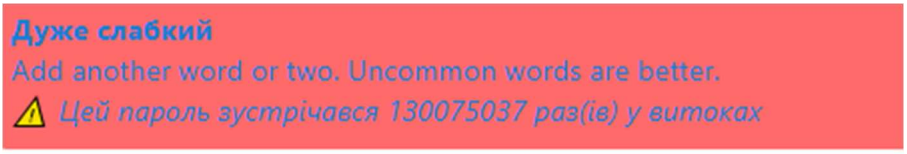


Рисунок 3.23 – Результат перевірки слабкого пароля

3.5 Тестування системи

Після реалізації функціоналу програмної системи було проведено етап тестування з метою перевірки її працездатності, стабільності та відповідності заявленим вимогам. Тестування виконувалося вручну на основі функціональних сценаріїв, що охоплюють усі ключові модулі: генератор паролів, перевірку складності, виявлення витоків (HIBP), інтерфейс користувача та перемикач теми.

Методологія тестування.

Тестування проводилось за методикою чорного ящика, при якій користувач взаємодіє з системою через вебінтерфейс без заглиблення у внутрішню реалізацію. Особливу увагу приділено:

- стабільності запитів до API;
- коректності обробки помилок;
- відображенню результатів на фронтенді;
- відповідності між введеними параметрами та отриманими паролями.

Сценарії ручного тестування.

Тест 1: Генерація паролів:

- умова: Обрати довжину 16, включити великі літери, цифри, спецсимволи;
- очікуваний результат: Виведення 6 різних паролів відповідної довжини;
- результат (рис. 3.24).

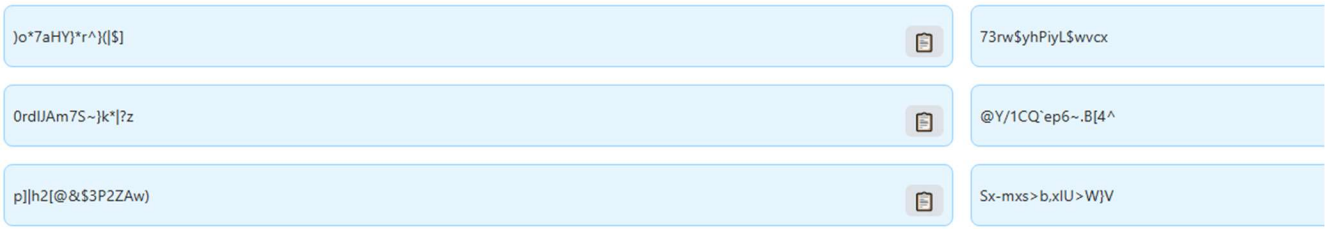


Рисунок 3.24 – Результат першого тесту

Тест 2: Легкозапам’ятовувані паролі:

- умова: Увімкнути опцію "запам’ятовуваний пароль";
- очікуваний результат: Пароль має шаблон: *Слово-ТваринаЧисло@*;

– результат (рис. 3.25).

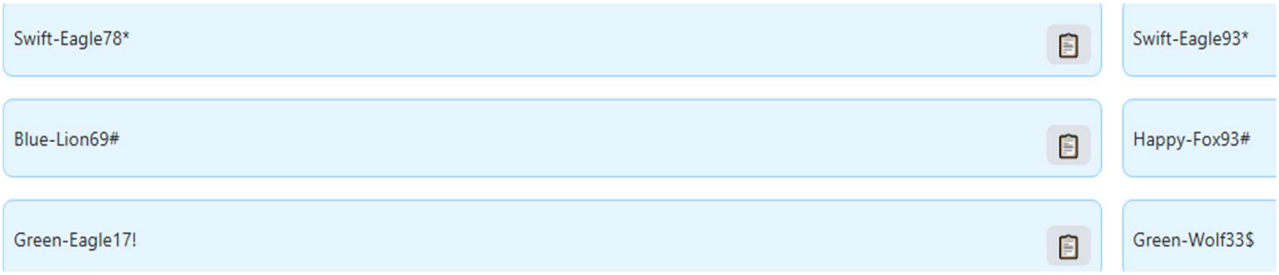


Рисунок 3.25 – Результат 2 тесту

Тест 3: Перевірка складності:

– умова: Ввести пароль 123456;

– очікуваний результат: Низька оцінка (0–1), рекомендації та повідомлення про наявність у витоках;

– результат (рис. 3.26).

Рисунок 3.26 – Результат 3 тесту

Тест 4: Сильний пароль:

– умова: Ввести пароль K9!wTy$4&jRe2023;

– очікуваний результат: Висока оцінка (3–4), відсутність витоків;

– результат (рис. 3.27).

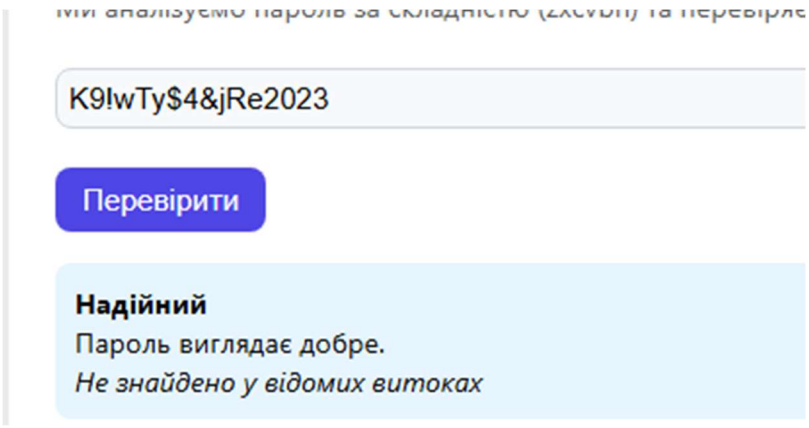


Рисунок 3.27 – Результат 4 тесту

Тест 5: Темна/світла тема:

– умова: Увімкнути перемикач теми;

– очікуваний результат: Зміна кольорів інтерфейсу, збереження працездатності;

– результат (рис. 3.28).

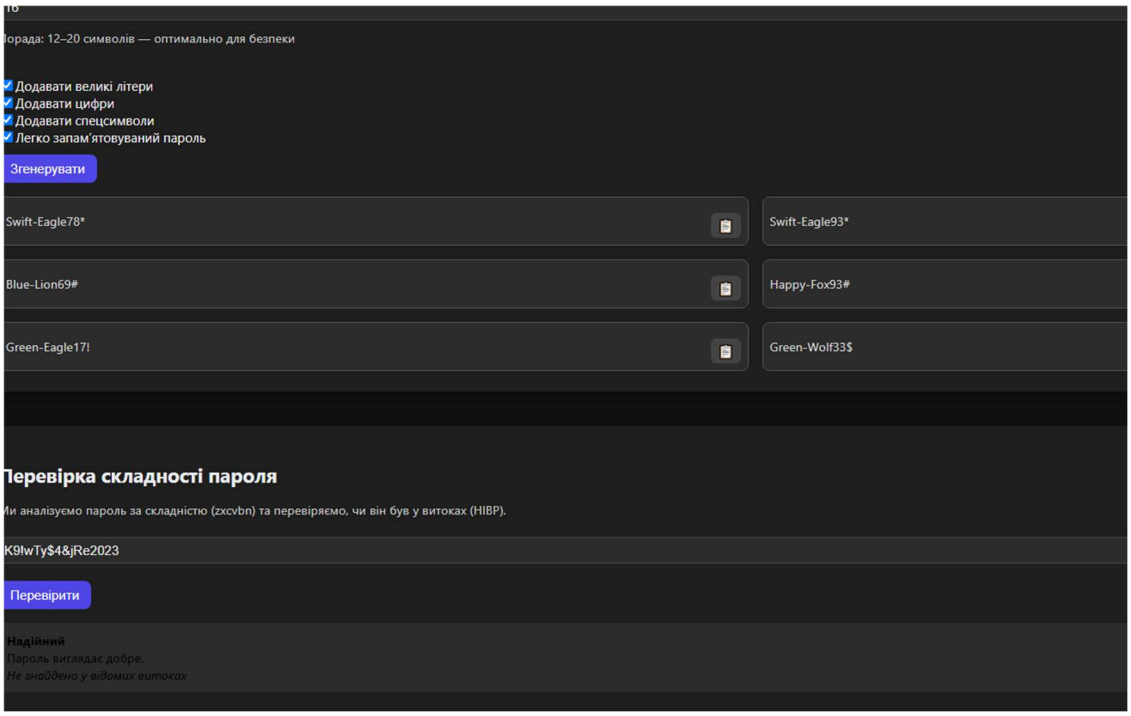
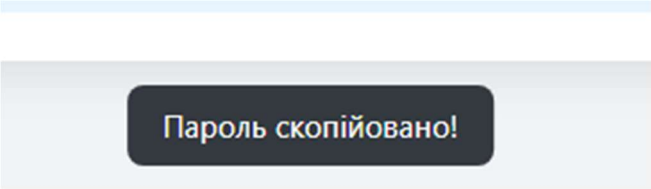


Рисунок 3.28 – Результат 5 тесту

Тест 6: Копіювання пароля:

- умова: Натиснути на іконку копіювання біля пароля;
- очікуваний результат: Вміст буфера змінено, відображено повідомлення;
- результат (рис. 3.29).



Рсиунок 3.29 – Результат 6 тесту

Висновки до розділу 3

У третьому розділі було здійснено повну реалізацію програмної системи для генерації та перевірки паролів, включаючи як серверну, так і клієнтську частину. Система побудована з урахуванням сучасних вимог до інформаційної безпеки, масштабованості та зручності використання.

У межах реалізації:

- розроблено генератор паролів, який підтримує налаштування довжини, включення символів різних типів, а також можливість створення легкозапам’ятовуваних паролів;
- імплементовано перевірку складності пароля з використанням бібліотеки zxcvbn, що дозволяє оцінювати не лише довжину, а й наявність шаблонів, послідовностей та слабких структур;
- реалізовано перевірку на наявність у витоках з допомогою сервісу Have I Been Pwned (HIBP), що значно підвищує реальну практичну цінність системи;
- створено зручний інтерфейс користувача, адаптований під мобільні пристрої, із підтримкою перемикача теми, локалізації, кнопками для копіювання паролів і миттєвим зворотним зв’язком;

− проведено повноцінне тестування функціоналу, яке підтвердило коректну роботу системи при виконанні основних сценаріїв, включаючи нестандартні ситуації.

Загалом, реалізована система демонструє ефективність поєднання практичних технологій Python та FastAPI з простим і гнучким фронтендом, що дозволяє застосовувати її як у навчальних, так і в реальних умовах, наприклад, для інтеграції у корпоративні системи або як основу для майбутніх розширень.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було розроблено програмну систему для генерації та перевірки паролів, яка вирішує актуальну проблему – забезпечення надійного захисту облікових записів користувачів від несанкціонованого доступу.

У роботі проведено:

− аналітичний огляд сучасних загроз, пов’язаних із використанням слабких паролів;

− аналіз найбільш поширених атак на паролі (перебір, словникові атаки, соціальна інженерія);

− обґрунтування вибору технологій для побудови надійної та зручної системи.

В результаті було реалізовано вебзастосунок, що включає такі ключові функціональні компоненти:

− генератор паролів з можливістю налаштування параметрів (довжина, великі літери, цифри, спецсимволи, варіанти для легкого запам’ятовування);

− перевірку складності пароля на основі бібліотеки zxcvbn, що дозволяє користувачеві отримати зворотний зв’язок і рекомендації щодо підвищення стійкості пароля;

− інтеграцію з сервісом Have I Been Pwned (HIBP) для перевірки, чи був введений пароль скомпрометований у публічних витоках;

− інтерфейс користувача, адаптивний для різних пристроїв, з підтримкою темної теми, локалізації та функціями швидкого копіювання згенерованих паролів.

Тестування системи показало, що всі модулі функціонують стабільно, демонструючи швидку обробку запитів та коректне реагування на різні сценарії взаємодії. Реалізований застосунок може бути використаний як самостійний сервіс, так і як основа для інтеграції в більш масштабні системи захисту інформації.

Отже, поставлені у роботі мета та завдання були успішно досягнуті, а отримані результати підтверджують практичну значущість розробленої системи в контексті підвищення кібербезпеки користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Python Software Foundation. Python 3.11 Documentation [Електронний ресурс]. – URL: https://docs.python.org/3/
2. FastAPI: modern, fast web framework for building APIs [Електронний ресурс]. – URL: https://fastapi.tiangolo.com/
3. Dropbox. zxcvbn password strength estimator [Електронний ресурс]. – URL: https://github.com/dropbox/zxcvbn
4. OWASP Foundation. Password Storage Cheat Sheet [Електронний ресурс]. – URL:https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
5. Hunt T. Have I Been Pwned: Pwned Passwords API v3 [Електронний ресурс]. – URL: https://haveibeenpwned.com/API/v3
6. Mozilla Developer Network (MDN). JavaScript Fetch API [Електронний ресурс]. – URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
7. Chart.js: Simple yet flexible JavaScript charting [Електронний ресурс]. – URL: https://www.chartjs.org/
8. Tailwind Labs. TailwindCSS Documentation [Електронний ресурс]. – URL: https://tailwindcss.com/docs
9. bcrypt: Password hashing for Python [Електронний ресурс]. – URL: https://pypi.org/project/bcrypt/
10. Passlib: Comprehensive password hashing framework [Електронний ресурс]. – URL: https://passlib.readthedocs.io/en/stable/
11. Коноваленко А. І. Захист інформації в комп’ютерних системах. – К.: КНУ, 2020. – 285 с.
12. ДСТУ 8302:2015. Бібліографічне посилання. Загальні положення та правила складання.
13. ДСТУ 3634-99. Захист інформації. Терміни та визначення.
14. Іващенко А. М. Основи інформаційної безпеки: навч. посіб. – Львів: ЛНУ, 2018. – 160 с.

15. Bulychev M. Secure password generators: overview and classification // Journal of Information Security. – 2020. – №3. – С. 47–53.
16. Schneier B. Secrets and Lies: Digital Security in a Networked World. – Wiley, 2015. – 432 p.
17. Sanderson A. Practical Cryptography for Developers. – O'Reilly Media, 2021. – 276 p.
18. Ferraiolo D., Kuhn D. Role-Based Access Control. – Artech House, 2003. – 312 p.
19. Python Requests Library [Електронний ресурс]. – URL: https://requests.readthedocs.io/
20. JetBrains. PyCharm IDE [Електронний ресурс]. – URL: https://www.jetbrains.com/pycharm/
21. Stack Overflow Developer Survey 2023 [Електронний ресурс]. – URL: https://survey.stackoverflow.co/2023/
22. Ristic I. SSL Labs: SSL/TLS Best Practices [Електронний ресурс]. – URL: https://www.ssllabs.com/
23. OWASP Top 10 – Application Security Risks [Електронний ресурс]. – URL: https://owasp.org/www-project-top-ten/
24. ISO/IEC 27001:2022. Information Security Management Systems – Requirements.
25. RFC 7617: HTTP Basic Authentication Scheme [Електронний ресурс]. – URL: https://tools.ietf.org/html/rfc7617
26. RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3 [Електронний ресурс]. – URL: https://tools.ietf.org/html/rfc8446
27. Brown D. Best Practices for Modern Web Application Security. – InfoQ, 2022. – 74 p.
28. Bezdushna H. Системи управління інформаційною безпекою. – Х.: ХНУРЕ, 2019. – 198 с.
29. Google Developers. Web Fundamentals – Secure Authentication [Електронний ресурс]. – URL: https://developers.google.com/web/fundamentals/security

30.Козленко І. Ю. Інформаційна безпека: сучасні загрози та методи захисту. – Київ: НТУУ «КПІ», 2021. – 137 с.

ДОДАТОК А

Екранні форми результат роботи програми

generator.py:

```
from fastapi import APIRouter
from pydantic import BaseModel
import random
import string

router = APIRouter()


adjectives = ["Red", "Blue", "Green", "Fast", "Bright", "Happy", "Dark", "Swift"]
animals = ["Fox", "Tiger", "Wolf", "Bear", "Lion", "Eagle", "Shark", "Panther"]
specials = "!@#$%&*"

class PasswordParams(BaseModel):
    length: int = 12
    use_upper: bool = True
    use_digits: bool = True
    use_specials: bool = True
    easy_to_remember: bool = False

def generate_easy_password(use_digits=True, use_specials=True):
    word = f"{random.choice(adjectives)}-{random.choice(animals)}"
    number = str(random.randint(10, 99)) if use_digits else ""
    special = random.choice(specials) if use_specials else ""
    return f"{word}{number}{special}"

def generate_strong_password(length, use_upper, use_digits, use_specials):
    charset = string.ascii_lowercase
    if use_upper:
        charset += string.ascii_uppercase
    if use_digits:
        charset += string.digits
    if use_specials:
        charset += string.punctuation
    return ''.join(random.choice(charset) for _ in range(length))

@router.post("/")
def generate_password(params: PasswordParams):
    if params.easy_to_remember:
        return {"password": generate_easy_password(params.use_digits,
params.use_specials)}
    else:
        return {"password": generate_strong_password(
            params.length,
            params.use_upper,
            params.use_digits,
            params.use_specials
        )}
```

main.py:

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
```

```
from backend.generator import router as generator_router
from backend.strength import router as strength_router

app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
app.include_router(generator_router, prefix="/generate")
app.include_router(strength_router, prefix="/strength")
```

schemas.py:

```
from pydantic import BaseModel

class PasswordParams(BaseModel):
    length: int = 12
    use_upper: bool = True
    use_digits: bool = True
    use_specials: bool = True
    easy_to_remember: bool = False
```

strength.py:

```
from fastapi import APIRouter
from pydantic import BaseModel
from zxcvbn import zxcvbn
import hashlib
import requests

router = APIRouter()

class PasswordCheckRequest(BaseModel):
    password: str

def check_password_pwned(password: str) -> str:
    sha1_password = hashlib.sha1(password.encode('utf-8')).hexdigest().upper()
    prefix = sha1_password[:5]
    suffix = sha1_password[5:]

    url = f"https://api.pwnedpasswords.com/range/{prefix}"

    try:
        response = requests.get(url, timeout=5)
        if response.status_code != 200:
            return "⚠️ Помилка перевірки через HIBP"

        for line in response.text.splitlines():
            hash_suffix, count = line.strip().split(":")
            if hash_suffix == suffix:
                return f"⚠️ Цей пароль зустрічався {count} раз(ів) у витоках"
        return "✅ Пароль не знайдено у зламаних базах"
    except Exception:
        return "❌ Не вдалося з'єднатися з HIBP"
```

```
@router.post("/")
def check_password(data: PasswordCheckRequest):
    result = zxcvbn(data.password)
    score = result['score']
    feedback = result['feedback']['suggestions']
    pwned = check_password_pwned(data.password)
    return {
        "score": score,
        "feedback": feedback,
        "breached": pwned
    }
```

app.js:

```
const API_BASE = "http://localhost:8000";

const translations = {
  ua: {
    title: "Генератор і перевірка паролів",
    genTitle: "Генерація декількох паролів",
    genHint: "Виберіть параметри та отримайте кілька безпечних варіантів пароля на
вибір.",
    lengthLabel: "Довжина (макс. 65 символів):",
    tip: "Порада: 12–20 символів — оптимально для безпеки",
    upper: "Додавати великі літери",
    digits: "Додавати цифри",
    specials: "Додавати спецсимволи",
    easy: "Легко запам’ятовуваний пароль",
    generate: "Згенерувати",
    checkTitle: "Перевірка складності пароля",
    checkHint: "Ми аналізуємо пароль за складністю (zxcvbn) та перевіряємо, чи він
був у витоках (HIBP).",
    checkBtn: "Перевірити",
    placeholder: "Введіть свій пароль",
    copied: "Пароль скопійовано!",
    lang: "UA"
  },
  en: {
    title: "Password Generator & Checker",
    genTitle: "Generate Multiple Passwords",
    genHint: "Choose parameters and get several strong password options.",
    lengthLabel: "Length (max 65 characters):",
    tip: "Tip: 12–20 characters is best for safety",
    upper: "Include uppercase letters",
    digits: "Include digits",
    specials: "Include special characters",
    easy: "Easy-to-remember password",
    generate: "Generate",
    checkTitle: "Password Strength Checker",
    checkHint: "We check your password strength (zxcvbn) and whether it has
appeared in leaks (HIBP).",
    checkBtn: "Check",
    placeholder: "Enter your password",
    copied: "Password copied!",
    lang: "EN"
  }
};

function applyLanguage(lang = "ua") {
  const t = translations[lang];
  document.querySelector("h1").textContent = t.title;
```

```
  document.querySelector("h2").textContent = t.genTitle;
  document.querySelectorAll("h2")[1].textContent = t.checkTitle;
  document.querySelector(".hint").textContent = t.genHint;
  document.querySelectorAll(".hint")[1].textContent = t.tip;
  document.getElementById("length").previousSibling.textContent = t.lengthLabel;
  document.querySelectorAll("label")[1].lastChild.textContent = t.upper;
  document.querySelectorAll("label")[2].lastChild.textContent = t.digits;
  document.querySelectorAll("label")[3].lastChild.textContent = t.specials;
  document.querySelectorAll("label")[4].lastChild.textContent = t.easy;
  document.getElementById("generateBtn").textContent = t.generate;
  document.querySelectorAll(".hint")[2].textContent = t.checkHint;
  document.getElementById("checkBtn").textContent = t.checkBtn;
  document.getElementById("checkInput").placeholder = t.placeholder;
  document.getElementById("langToggle").textContent = t.lang;
}
document.getElementById("themeToggle").addEventListener("click", () => {
  document.body.classList.toggle("dark-mode");
  const isDark = document.body.classList.contains("dark-mode");
  document.getElementById("themeToggle").textContent = isDark ? "🌞" : "🌙";
});

document.getElementById("langToggle").addEventListener("click", () => {
  const current = localStorage.getItem("lang") || "ua";
  const newLang = current === "ua" ? "en" : "ua";
  localStorage.setItem("lang", newLang);
  applyLanguage(newLang);
});

window.addEventListener("DOMContentLoaded", () => {
  const savedLang = localStorage.getItem("lang") || "ua";
  applyLanguage(savedLang);
});

function showToast(message) {
  const toast = document.getElementById("toast");
  toast.textContent = message;
  toast.classList.add("show");
  setTimeout(() => {
    toast.classList.remove("show");
  }, 2000);
}

document.getElementById("generateBtn").addEventListener("click", async () => {
  const length = Math.min(parseInt(document.getElementById("length").value), 65);
  const use_upper = document.getElementById("upper").checked;
  const use_digits = document.getElementById("digits").checked;
  const use_specials = document.getElementById("specials").checked;
  const easy = document.getElementById("easy").checked;

  const passwordOptions = document.getElementById("passwordOptions");
  passwordOptions.innerHTML = "";
  for (let i = 0; i < 6; i++) {
    try {
      const res = await fetch(`${API_BASE}/generate/`, {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          length,
          use_upper,
          use_digits,
          use_specials,
          easy_to_remember: easy
```

```
        }),
      });

      const data = await res.json();
      const password = data.password || "Помилка";

      const div = document.createElement("div");
      div.classList.add("password-block");

      const passText = document.createElement("span");
      passText.textContent = password;

      const copyBtn = document.createElement("button");
      copyBtn.textContent = "📋";
      copyBtn.className = "copy-btn";
      copyBtn.onclick = () => {
        navigator.clipboard.writeText(password).then(() => {
          const lang = localStorage.getItem("lang") || "ua";
          showToast(translations[lang].copied);
        });
      };

      div.appendChild(passText);
      div.appendChild(copyBtn);
      passwordOptions.appendChild(div);
    } catch {
      const div = document.createElement("div");
      div.textContent = "Помилка генерації";
      passwordOptions.appendChild(div);
    }
  }
});

document.getElementById("checkBtn").addEventListener("click", async () => {
  const password = document.getElementById("checkInput").value.trim();
  const box = document.getElementById("strengthResult");
  const lang = localStorage.getItem("lang") || "ua";

  if (!password) {
    box.innerText = lang === "ua" ? "Введіть пароль!" : "Please enter a
password!";
    return;
  }

  try {
    const res = await fetch(`${API_BASE}/strength/`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ password }),
    });
    const data = await res.json();

    const strengthMessages = {
      ua: ["Дуже слабкий", "Слабкий", "Посередній", "Добрий", "Надійний"],
      en: ["Very weak", "Weak", "Medium", "Good", "Strong"]
    };

    const fallbackMsg = {
      ua: "Пароль виглядає добре.",
      en: "The password looks good."
    };
```

```
    let breach = "";
    if (data.breached?.error) {
      breach = lang === "ua"
        ? "Помилка з'єднання з базою витоків"
        : "Error connecting to breach database";
    } else if (data.breached?.breached) {
      breach = lang === "ua"
        ? `Знайдено у витоках (${data.breached.count} разів)`
        : `Found in breaches (${data.breached.count} times)`;
    } else {
      breach = lang === "ua"
        ? "Не знайдено у відомих витоках"
        : "Not found in known breaches";
    }
    const title = strengthMessages[lang][data.score];
    const feedback = data.feedback.length ? data.feedback.join("<br>") :
fallbackMsg[lang];

    box.innerHTML =
`<strong>${title}</strong><br>${feedback}<br><em>${breach}</em>`;
  } catch {
    box.innerText = lang === "ua"
      ? "Помилка перевірки пароля."
      : "Password check failed.";
  }
});
```

index.html:

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Система генерації паролів</title>
  <link rel="stylesheet" href="styles.css" />
</head>
<body>
  <button id="themeToggle" class="theme-toggle">🌙</button>
  <button id="langToggle" class="lang-toggle">UA</button>

  <main class="container">
    <header>
      <h1>Генератор і перевірка паролів</h1>
    </header>

    <!-- Генерація -->
    <section>
      <h2>Генерація декількох паролів</h2>
      <p class="hint">Виберіть параметри та отримайте кілька безпечних варіантів
пароля на вибір.</p>
      <label>Довжина (макс. 65 символів):
        <input type="number" id="length" value="12" min="4" max="65">
      </label>
      <small class="hint">Порада: 12–20 символів — оптимально для
безпеки</small><br>
      <label><input type="checkbox" id="upper" checked> Додавати великі
літери</label><br>
      <label><input type="checkbox" id="digits" checked> Додавати
цифри</label><br>
```

```
      <label><input type="checkbox" id="specials" checked> Додавати
спецсимволи</label><br>
      <label><input type="checkbox" id="easy"> Легко запам’ятовуваний
пароль</label><br>
      <button id="generateBtn">Згенерувати</button>

      <div id="passwordOptions" class="generated-list"></div>
    </section>

    <!-- Перевірка -->
    <section>
      <h2>Перевірка складності пароля</h2>
      <p class="hint">Ми аналізуємо пароль за складністю (zxcvbn) та перевіряємо,
чи він був у витоках (HIBP).</p>
      <input type="text" id="checkInput" placeholder="Введіть свій пароль">
      <button id="checkBtn">Перевірити</button>
      <div id="strengthResult"></div>
    </section>
  </main>

  <div id="toast" class="toast"></div>

  <script src="app.js"></script>
</body>
</html>
```

styles.css:

```
body {
  font-family: "Segoe UI", sans-serif;
  font-size: 15px;
  margin: 0;
  padding: 0;
  background-color: #f1f3f5;
  color: #212529;
  transition: background-color 0.3s, color 0.3s;
}

.container {
  max-width: 100%;
  padding: 2rem 3rem;
  box-sizing: border-box;
}

header, section {
  margin-bottom: 2.5rem;
  background-color: #ffffff;
  padding: 1.5rem;
  border-radius: 12px;
  box-shadow: 0 8px 24px rgba(0, 0, 0, 0.08);
}

h1 {
  text-align: center;
  font-size: 2rem;
  margin-bottom: 1rem;
}

h2 {
  margin-bottom: 0.5rem;
}
```

```
.hint {
  font-size: 0.9rem;
  color: #555;
  margin-bottom: 1rem;
  display: block;
}

input[type="text"],
input[type="number"] {
  width: 100%;
  padding: 6px;
  margin-top: 6px;
  margin-bottom: 10px;
  border: 1px solid #ced4da;
  border-radius: 8px;
  background-color: #f8f9fa;
  font-size: 0.95rem;
}

button {
  padding: 8px 14px;
  background-color: #4f46e5;
  color: white;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  margin-top: 10px;
  font-size: 0.95rem;
}

button:hover {
  background-color: #4338ca;
}

#passwordOptions {
  margin-top: 1rem;
  display: flex;
  flex-wrap: wrap;
  gap: 1rem;
}

.password-block {
  display: flex;
  justify-content: space-between;
  align-items: center;
  gap: 8px;
  padding: 8px;
  background-color: #e7f5ff;
  border-radius: 8px;
  border: 1px solid #91d1ff;
  flex: 1 1 calc(33% - 1rem);
  word-break: break-word;
  font-size: 0.9rem;
}

.copy-btn {
  background-color: #dee2e6;
  border: none;
  border-radius: 6px;
  padding: 4px 8px;
  cursor: pointer;
```

```
  font-size: 1rem;
  transition: background-color 0.2s;
}

.copy-btn:hover {
  background-color: #ced4da;
}

#strengthResult {
  margin-top: 1rem;
  padding: 10px;
  border-radius: 8px;
  background-color: #e7f5ff;
  color: #000000;
  font-size: 0.9rem;
}

/* Темна тема */
body.dark-mode {
  background-color: #121212;
  color: #f1f1f1;
}

body.dark-mode header,
body.dark-mode section {
  background-color: #1e1e1e;
  box-shadow: 0 8px 24px rgba(0, 0, 0, 0.4);
}

body.dark-mode input {
  background-color: #2c2c2c;
  color: #f1f1f1;
  border: 1px solid #444;
}

body.dark-mode #strengthResult {
  background-color: #2c2c2c;
  color: #000000;
}

body.dark-mode .password-block {
  background-color: #2c2c2c;
  color: #ddd;
  border-color: #555;
}
body.dark-mode .hint {
  color: #ccc;
}

body.dark-mode .copy-btn {
  background-color: #444;
  color: #fff;
}

body.dark-mode .copy-btn:hover {
  background-color: #666;
}

body.dark-mode h1,
body.dark-mode h2 {
  color: #e9ecef;
}
```

```
.theme-toggle {
  position: fixed;
  top: 15px;
  right: 15px;
  background: #444;
  color: white;
  border: none;
  border-radius: 6px;
  padding: 8px 14px;
  cursor: pointer;
}
.toast {
  position: fixed;
  bottom: 20px;
  left: 50%;
  transform: translateX(-50%);
  background-color: #343a40;
  color: #fff;
  padding: 10px 18px;
  border-radius: 8px;
  font-size: 0.95rem;
  opacity: 0;
  pointer-events: none;
  transition: opacity 0.3s ease-in-out;
  z-index: 1000;
}

.toast.show {
  opacity: 1;
}

body.dark-mode .toast {
  background-color: #222;
  color: #ddd;
}
.lang-toggle {
  position: fixed;
  top: 15px;
  right: 70px;
  background: #444;
  color: white;
  border: none;
  border-radius: 6px;
  padding: 8px 14px;
  cursor: pointer;
}
```

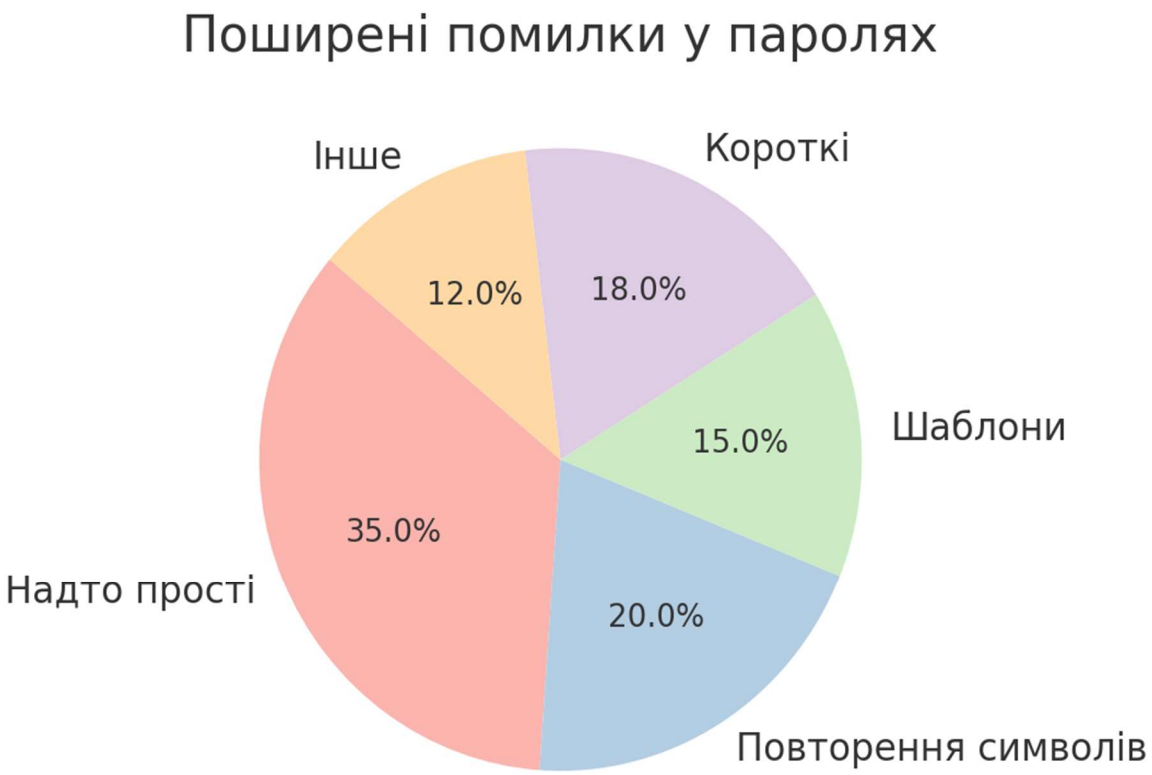
ДОДАТОК Б

Діаграми до розділу 3

Порівняння складності паролів

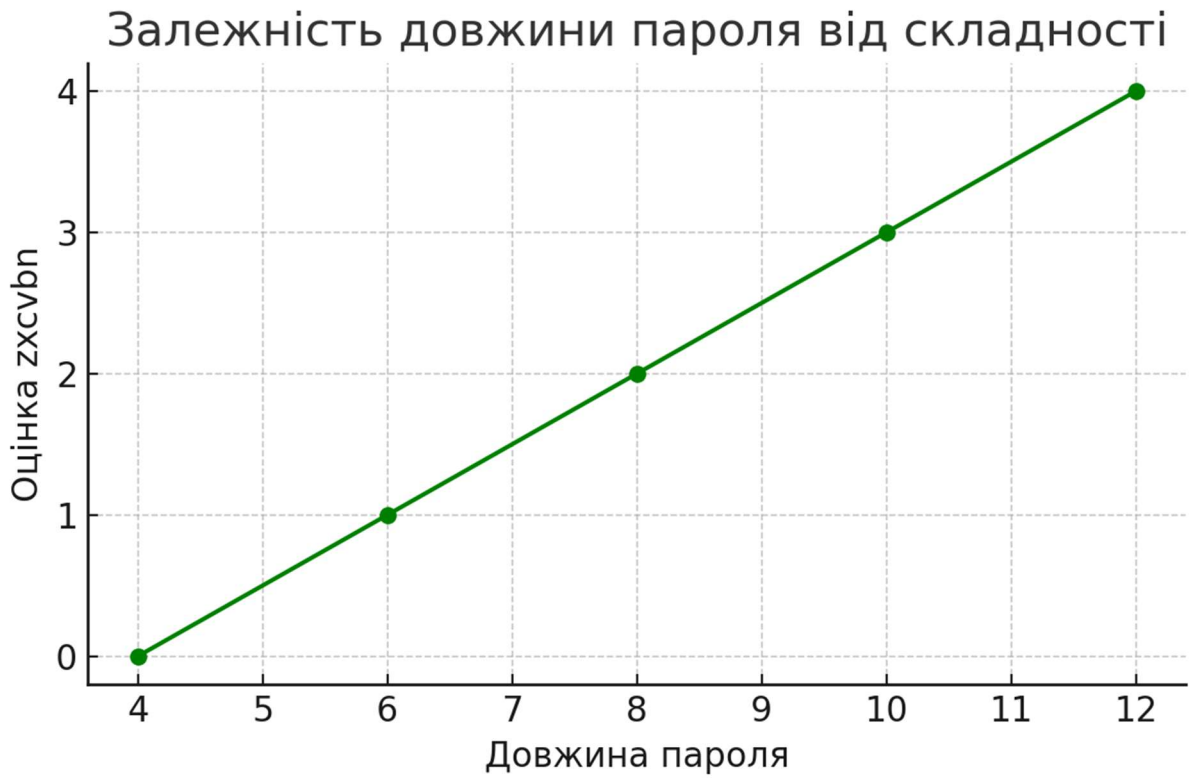
Діаграма ілюструє оцінку складності (0–4) різних типів паролів за результатами аналізу бібліотекою zxcvbn.

Поширені помилки у створенні паролів



На круговій діаграмі представлено найбільш поширені помилки, яких припускаються користувачі при створенні паролів.

Залежність довжини пароля від рівня складності



На графіку показано, як збільшення довжини пароля впливає на його оцінку складності за версією zxcvbn.